Design of the Gas-Puff Imaging Diagnostic for Wendelstein 7-X

by

Kevin Tang

Submitted to the Department of Nuclear Science and Engineering
in partial fulfillment of the requirements for the degree of

Bachelor of Science in Nuclear Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author .....
Department of Nuclear Science and Engineering
May 16, 2019

Certified by .....
James L. Terry
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Michael P. Short
Chairman, Department Committee for Undergraduate Students

Design of the Gas-Puff Imaging Diagnostic for Wendelstein 7-X

by

Kevin Tang

Submitted to the Department of Nuclear Science and Engineering
on May 16, 2019, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Nuclear Science and Engineering

Abstract

Stellarators, being not as well-studied as tokamaks, have plenty of interesting physics to examine, as investigations of stellarators as a viable configuration for future power plants continue. One of these aspects is boundary turbulence in the plasma, as the magnetic configuration in stellarators is different from that in tokamaks and thus provides different plasma behavior. To study this turbulence, we are designing a "gas-puff imaging" diagnostic to install onto the Max Planck Institute of Plasma Physics's Wendelstein 7-X (W7-X), which is currently the world's most advanced and largest stellarator. This diagnostic employs a fast-camera to observe a localized puff of gas as it interacts with the boundary plasma near the last closed flux surface of the plasma. The diagnostic consists of a fast-camera component, a light-collection component, a "gas-puff" component with valves to inject controlled amounts of gas, and a component for valve control and data collection purposes. This thesis documents some of the aspects of the design of the diagnostic and its components for W7-X.

Thesis Supervisor: James L. Terry
Title: Principal Research Scientist

Acknowledgments

I'd like to thank James L. Terry, Seung-Gyou Baek, and Sean Ballinger for their mentorship and assistance over the past two years.

Contents

1 Introduction **15**

2 Background **17**

2.1 Differences between Tokamaks and Stellarators 17

2.2 Phantom Fast-Camera Diagnostic Overview 19

2.3 Gas-Puff Imaging Diagnostic Standard 21

3 Design Methodology and Results **23**

3.1 Port Selection 23

3.2 Fast-Camera Subsystem 24

3.2.1 Simulations for the Phantom v710 Optical Setup 26

3.2.2 Simulations for the APDCAM-10G Optical Setup 27

3.3 Gas-Puff Valve Subsystem 29

3.4 Valve Control and Data Collection Subsystem 31

3.4.1 Red Pitaya Field-Programmable Gate Array 31

3.4.2 Python Graphical User Interface 34

3.5 Heat Loading Simulations 36

3.5.1 Heat Load Analysis of the Gas Puff Nozzle 36

3.5.2 Heat Load Analysis of the Immersion Tube 38

4 Summary **41**

A Figures **45**

B Tables **55**

C Code and Scripts **59**

C.1 block_design.tcl 59

C.2 config.yml 64

C.3 extension_connector.xdc 66

C.4 GPI_2.hpp 67

C.5 GPI_2.py 75

C.6 DataCollect.vhd 79

C.7 outputs.vhd 81

C.8 GPI_Valve_GUI.py 87

List of Figures

2-1 Schematics of magnet and plasma configurations in (a) tokamaks and (b) stellarators [1]. 18

2-2 Bench view of the Phantom diagnostic within its soft iron box. The Phantom camera, seen on the top rail, is optically connected to a turning mirror, then to a beamsplitter, and finally to a lens coupled with the fiber optic bundle. 19

2-3 (Left) View of the MIT fast-camera with divertor panels labeled and a cross-section of plasma flux surfaces superimposed. (Right) A single-frame snapshot of plasma emission, showing the typical enhanced emission along the divertor strike-line. 20

2-4 Overview of the mechanics for gas-puff imaging. The neutral gas is puffed perpendicular to the local magnetic field, and the line emissions are then observed by the fast-camera pointing parallel to the magnetic field at the point of interaction to observe boundary turbulence [2]. . 21

3-1 Fusion Instruments APDCAM-10G (left) and the microlens array (right). This will be the camera we will be using for the GPI diagnostic, with a 4x8 microlens array coupled to the camera for near 100% photon detection efficiency. 25

3-2 (Top) Optical system for the case where the Phantom v710 is chosen for the GPI. Photons from the plasma in the top right are reflected by the angled turning mirror, then channeled through the vacuum window, and finally through the series of lenses to the focus of the camera on the far left. (Bottom) Zoomed in view of the system of lenses and objective lens pictured in the far left of the overview. 26

3-3 Optical system for the case where the APDCAM-10G is chosen for the GPI. Photons from the plasma in the top right are reflected by the angled turning mirror, then channeled through the vacuum window, and through three lenses to focus onto the camera. 28

3-4 CAD rendering of the gas-puff valve subsystem with all of the components in place on the mounting plate. 29

3-5 General schematic of the Red Pitaya STEMlab 125-14 components. The Red Pitaya has on-board processor and FPGA, along with RAM and network connectivity on top of customizable pins in the extension connectors [3]. 32

3-6 Detailed pin descriptions of the extension connectors [4]. 32

3-7 General overview of the FPGA design layout. 33

3-8 Overview of the Python GUI. 35

3-9 COMSOL analysis of the gas puff nozzle with the back panel cooled. 36

3-10 COMSOL analysis of an idealized situation for the gas puff nozzle with a larger fraction of the mounting bracket cooled to 30°C. 37

3-11 COMSOL analysis of the optical system with one set of cooling pipes. Port AEE50 can be seen on the left, while the 880°C plasma sheet can be seen on the right. 38

3-12 COMSOL analysis of the optical system with two sets of cooling pipes flanking the vacuum window to cool the length of the tube as much as possible and therefore the window as well. 39

4-1 Overview of all of the components of the GPI diagnostic in place at their respective port locations. 41
4-2 Overview of the entire GPI diagnostic from a different angle than in Fig. 4-1. The sightline intersecting with the injected gas can be clearly seen. 42

A-1 Plasma cross section at the toroidal angle of the gas puff nozzle at port AEK50. Prominent plasma features such as the LCFS in blue and plasma islands in red are also shown. 46
A-2 Various details for general characteristics of the Zemax simulated Phantom v710 optical system. 47
A-3 Spot diagram for the Phantom v710 simulation. The image on the top left is that of the on-axis object, whereas the other two correspond to off-axis objects. We can thus see that the focused image spot is at best 0.4 mm in size and at worst 1.4 mm, which is acceptable for the 4x4 mm detector. 48
A-4 Geometric encircled energy plot for the Phantom v710 simulation. The on-axis case reaches 100% enclosed photon energy at around 0.34 mm, while the off-axis cases reach about 90% at that point. 49
A-5 Various details for general characteristics of the Zemax simulated APDCAM-10G optical system. 50
A-6 Spot diagram for the APDCAM-10G simulation. The image on the top left is that of the on-axis object, whereas the other two correspond to off-axis objects. We can thus see that the focused image spot is at best 0.1 mm in size and at worst 0.2 mm, which is well acceptable for the 23.4x39 mm detector. 51
A-7 Geometric encircled energy plot for the APDCAM-10G simulation. The on-axis case reaches 100% enclosed photon energy at around 0.055 mm, while the off-axis cases reach about 90% at that point. 52
A-8 Overview of the FPGA design in Vivado Design Suite. 53

List of Tables

B.1 Detailed specification of the various distances and parameters necessary to generate the optical system for the Phantom v710 case. The radius column refers to the radius of curvature of the optical elements, the thickness the distance separation between two faces of those elements, and the semi-diameter their radial size, all of which are in mm. The materials were selected to have the same optical qualities as those that would be actually used and blank materials indicate air. These values have been optimized within Zemax to provide the best focus at the detection plane. The overall distance between the gas puff and the detection plane is approximately 2800 mm. 56

B.2 Detailed specification of the various distances and parameters necessary to generate the optical system for the APDCAM-10G case. The columns are defined the same way as those in Table B.1. These values have been optimized within Zemax to provide the best focus at the detection plane. The overall distance between the gas puff and the detection plane is also approximately 2800 mm. 57

Chapter 1

Introduction

Since the 1960s, research with regards to harnessing nuclear fusion to generate energy has taken off and branched into multiple subfields. Within the magnetic confinement fusion subfield, the leading reactor design candidate for realizing a fusion nuclear power plant is the tokamak, since historically it has provided the best results in approaching net energy generation. However, recent manufacturing innovations and work with another type of magnetic configuration, the stellarator, have shown that it may be another promising candidate for energy generation purposes, as it is able to contain the fusion plasma in steady-state operation.

In the field of fusion research, while tokamaks have been widely studied, stellarators are relatively novel; thus there are many aspects of stellarator physics that require study. One particular issue that magnetic confinement fusion faces is plasma turbulence, which carries heat and particles out of the confined plasma. As such, plasma turbulence and its behavior must be mitigated in order to successfully implement a viable fusion power plant in the future.

Wendelstein 7-X (W7-X), of the Max Planck Institute for Plasma Physics (IPP) in Greifswald, Germany, is currently the world's largest and most advanced stellarator now undergoing experimental campaigns with hydrogen plasmas, with the long-term goal of attaining steady-state continuous operation for thirty minutes. An MIT Plasma Science and Fusion Center (PSFC) group, consisting of Principal Research Scientist James L. Terry, Research Scientist Seung-Gyou Baek, PhD Student

Sean Ballinger, and me, is currently designing a gas-puff imaging (GPI) diagnostic to install onto W7-X in late 2019 or early 2020, in time for W7-X's second operation phase in 2021. Although GPI diagnostics have been previously installed onto tokamaks, the GPI diagnostic being designed for installation onto W7-X will be the first of its kind for a stellarator [5] [6]. In using a fast-camera to observe the line emissions from inert gas interactions between the locally puffed gas and the outer edge of the plasma near the last closed flux surface (LCFS), some of the characteristics of boundary turbulence are measured by the GPI [7]. This diagnostic contains a fast-camera system, a light-collection system, a gas-puffing system, and a system for control and data collection purposes.

The purpose of this thesis is to document my contributions to the process of designing the GPI diagnostic for W7-X.

Chapter 2

Background

2.1 Differences between Tokamaks and Stellarators

Many of the differences between tokamaks and stellarators arise from their geometries, which determine their magnetic configurations and other attributes from these magnetic field differences [1]. Over the past 60 years, much research in the field of magnetic confinement fusion has been done on tokamaks and this research has led to the construction of the experimental test reactor, ITER. ITER, being built today, will operate with deuterium-tritium fuel in hopes of achieving a sustained energy "gain" of approximately 10 [8]. However, on the stellarator side of the field, Wendelstein 7-X is the most advanced, with its recent successful operation being an indicator that stellarators could prove to be an alternate model for a power plant [9].

Although both tokamaks and stellarators are toroidal plasma confinement systems using magnetic fields, the method by which confinement is achieved is quite different. Both designs need to perform a rotational transform of the toroidal magnetic field to mitigate curvature drift and to have an equilibrium between plasma kinetic pressure and magnetic pressure [1]. There are three ways of producing this twist in the magnetic field: "(i) creating a poloidal field by a toroidal electric current; (ii) rotating the poloidal cross-section of stretched flux surfaces around the torus; (iii) making the magnetic axis non-planar" [1]. Tokamaks use the first method by producing a large toroidal plasma current, whereas stellarators rely upon the latter two methods using

external non-axisymmetric magnet coils, as shown in Fig. 2-1, where (a) is a tokamak and (b) is a stellarator design.

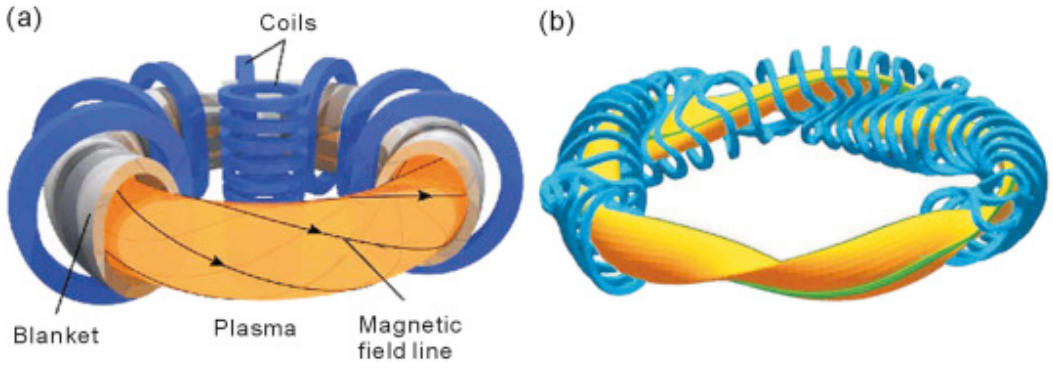

Figure 2-1: Schematics of magnet and plasma configurations in (a) tokamaks and (b) stellarators [1].

From their designs, tokamaks can confine all collisionless particles but typically generate the toroidal current through a transformer, which makes them difficult to operate in steady-state, while stellarators are inherently current free and able to operate in steady-state but have more unconfined particle orbits [1]. One of the differences resulting from the different magnetic configurations is different magnetohydrodynamic (MHD) instabilities. In tokamaks, the toroidal plasma current can drive macroscopic and microscopic MHD instability, whereas in stellarators, the lack of the toroidal plasma current removes current-driven MHD instabilities [1]. The large plasma current is also the source of "disruptions" in tokamaks, where the plasma can suddenly collapse, which is dangerous to the reactor as the excess energy within the plasma is released with little notice into supporting structures. Because these major disruptions do not exist in stellarators, this problem is not a concern for stellarators.

However, despite having no net toroidal current, two forms of small plasma currents do exist in stellarators simply because they are toroidal systems, though the currents are not large enough to destabilize the plasma through MHD modes. One of those is the bootstrap current caused by pressure gradients at low collisionality and the other is the "parallel Pfirsch-Schluter current used to compensate the non-zero poloidal plasma current", both of which are also present in tokamaks [1].

As such, while there are major differences in turbulent behavior between tokamaks and stellarators, there are also microscopic similarities. Investigating these aspects will provide new insights on plasma turbulence in different reactor configurations.

2.2 Phantom Fast-Camera Diagnostic Overview

As a precursor to fielding the GPI diagnostic, two years ago our group installed a fast-camera diagnostic onto W7-X. This is a far simpler installation compared to a GPI system. Essentially, one using a fast-framing camera with a view of the plasma to monitor, on turbulence time-scales, the visible light that is emitted by the boundary plasma. The fast-framing camera that we utilized is the Phantom v710, capable of taking up to 391,000 frames per second with 64x64 pixel resolution. This camera is housed in a soft iron box and utilizes a turning mirror and a beamsplitter, as its view had to be shared with another much slower, machine-protection camera. Both cameras' views are then provided by a lens coupled to a coherent fiber optic image-guide with a toroidal view of the full plasma cross-section. A photo of this setup for the Phantom is shown in Fig. 2-2, although the slow camera is not shown.

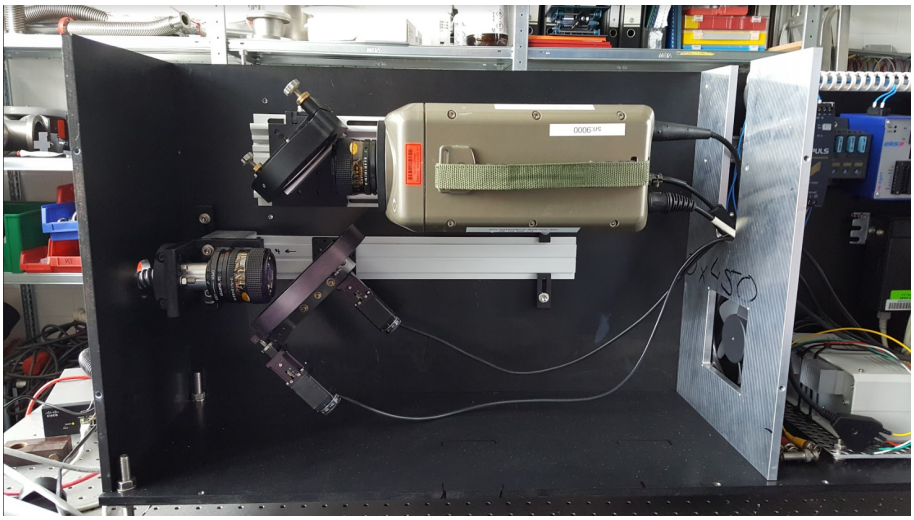

Figure 2-2: Bench view of the Phantom diagnostic within its soft iron box. The Phantom camera, seen on the top rail, is optically connected to a turning mirror, then to a beamsplitter, and finally to a lens coupled with the fiber optic bundle.

The diagnostic was installed at the W7-X port labeled AEQ30. The reconstructed CAD view, along with a cross-section of the plasma flux surfaces, is shown in Fig. 2-3. A single frame of the emission from the plasma is shown at the right. The curved line of bright emission in the top right of that frame is the "strike-line", where the plasma makes contact with the divertor target-plate.

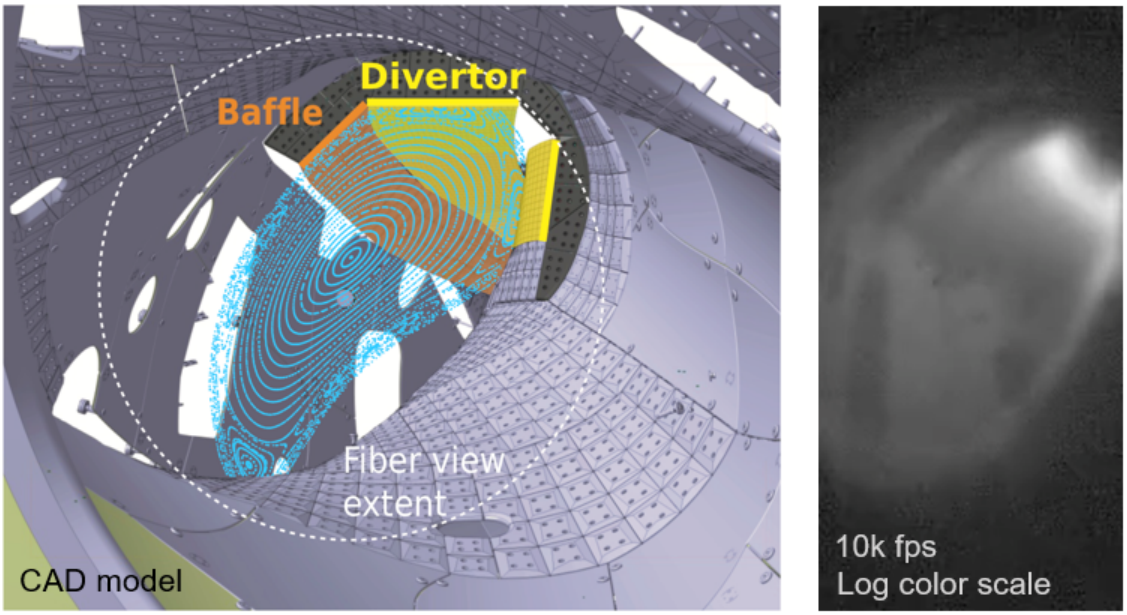

Figure 2-3: (Left) View of the MIT fast-camera with divertor panels labeled and a cross-section of plasma flux surfaces superimposed. (Right) A single-frame snapshot of plasma emission, showing the typical enhanced emission along the divertor strike-line.

From this view, visible light from the cooler regions of the plasma—the boundary and the divertor regions—is emitted and imaged by the camera, allowing plasma fluctuations in those regions to be observed.

I was present on-site at W7-X during the installation of this diagnostic; as such, I aided in the process as well. My role in the installation of the Phantom was primarily to align the image-guide when we coupled it to the optical system that directed the output of the image-guide to the two cameras. The image-guide was aligned on the end pointing into the plasma vessel but needed alignment on the end connecting to the diagnostic. Because the fibers in the image-guide were grouped together in a

rectangular format, alignment of the image-guide entailed illuminating that face of the image-guide and rotating the bundle to match the desired angle for the rectangle, and then pointing the guide in order to achieve the view as shown in Fig. 2-3 (Right).

2.3 Gas-Puff Imaging Diagnostic Standard

Gas-puff imaging diagnostics have been implemented on tokamak experiments such as the National Spherical Torus Experiment (NSTX) and MIT's Alcator C-Mod [7]. Both diagnostics on these experiments were similarly structured, with a controlled neutral gas-puff, typically helium or deuterium, being injected at the edge of the plasma [2]. The visible light emission from the gas cloud was then imaged with a fast-camera in short exposures, "shorter than the autocorrelation time of the turbulence", where the images obtained in the radial vs. poloidal plane corresponded directly to the instantaneous structure of the turbulence and followed the motion of the turbulence [7]. A schematic overview of this concept is shown in Fig. 2-4.

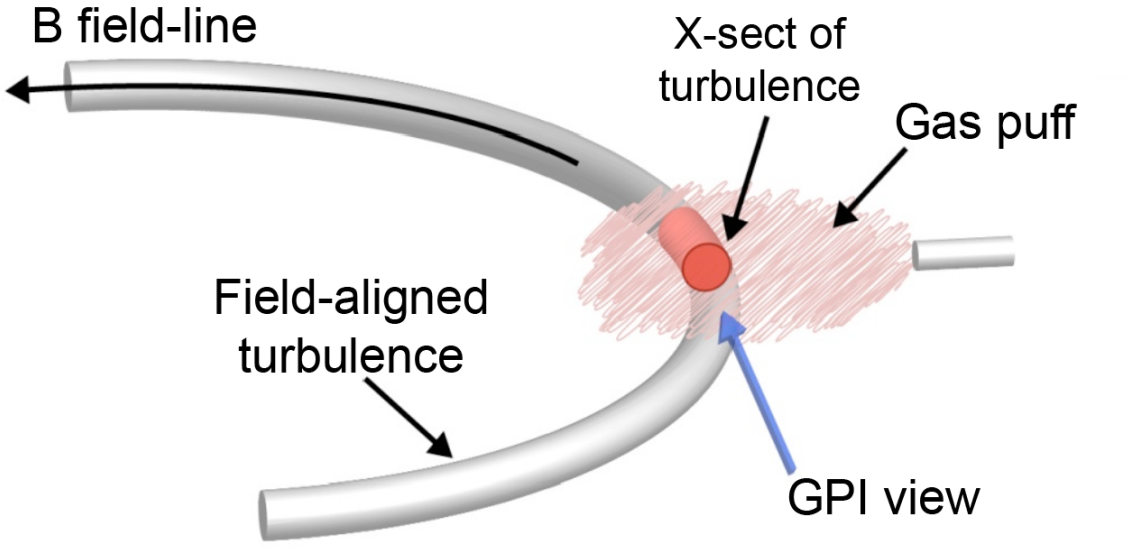

Figure 2-4: Overview of the mechanics for gas-puff imaging. The neutral gas is puffed perpendicular to the local magnetic field, and the line emissions are then observed by the fast-camera pointing parallel to the magnetic field at the point of interaction to observe boundary turbulence [2].

Our GPI diagnostic implementation follows the standards set by previous implementations of the diagnostic, also containing a fast-camera component with framing rates faster than the lifetime of turbulence in the plasma. This fast-camera component will be coupled with a gas-puffing component in the sight-lines of the collecting optics. The localized gas puff, either helium or deuterium, will interact with the boundary plasma, exciting line emissions from the puffed-gas cloud. We use the 652 nm Hα line for H_2 gas and the 587 nm line from neutral helium when He gas is puffed, both of which are the strongest lines in the visible spectrum for those elements. The emitted light is collected in an appropriate geometry using high-throughput optics, which then present a two-dimensional image of the emission to the camera system. Because the light emission responds to fluctuations in the local electron density and temperature, imaging the light fluctuations acts as a proxy for imaging the local plasma fluctuations [7]. As such, turbulent behavior at the plasma boundary is captured by the camera for further analysis.

Chapter 3

Design Methodology and Results

Because the GPI diagnostic contains multiple components and systems, each individual portion was selected by screening through different criteria and comparing with various available options. The major design portions were selection of the port to house the GPI, choosing which camera to use for the fast-camera subsystem, the design of the gas injection subsystem, the design of the valve control and data collection subsystem, and various simulations and tests leading to the Conceptual Design Review (CDR) held at W7-X in April 2019. Information regarding the specifications needed for W7-X and regarding aspects of W7-X itself was provided to us by Olaf Grulke, Adrian von Stechow, and Christoph von Sehren.

3.1 Port Selection

The first priority of this diagnostic was to choose an appropriate port as its host, which ideally would have the correct distance for our optical focus as well as provide us with a good view near the last closed flux surface that would interact with the puffed gas. Work to determine the ideal port began two years ago in the summer of 2017, where we conducted sightline analyses for all available ports on W7-X, and then narrowed down further through these following constraints:

(1) The sightline should be approximately aligned with the local magnetic field at the location where the viewing chord intersects the gas puff;

(2) The view should be near the plasma boundary, i.e. within 0.05 m of the LCFS;

(3) The gas puff must not be too far from the viewing port since we must maximize the signal to noise for the detected line emission from the puffed gas;

(4) It is optimal if the sightline is perpendicular to the local normal to the flux-surface at the location where the sightline intersects the gas puff;

(5) Ideally the region viewed should be a region of interest with expected turbulence.

Following from initial work done by Seung-Gyou Baek, where all possible ports with all possible views were considered, I subsequently narrowed down the selection of possible ports by collapsing the views to an average sightline and investigating the actual availability of ports. After further looking at CAD renderings of the plasma vessel and realizing that many ports were occupied for heating purposes or for other diagnostics, we eventually were able to consolidate the selections to five ports that would still satisfy the above conditions, albeit through use of a turning mirror to ensure the correct view from the first two conditions.

The five possible ports were AEE50, AEE11, AEK41, AEK10, and AEK50. Upon closer inspection and consultation with Olaf Grulke, we determined that port AEE50 would be appropriate for our purposes, with the view directed towards AEK50. The expected plasma cross-sectional view is shown in Appendix Fig. A-1. Port AEE50 will house the light-collecting optics for the fast-camera subsystem, while port AEK50 will host the gas-supplying capillary and the nozzle to puff the gas into the vessel near the plasma boundary.

3.2 Fast-Camera Subsystem

For the fast-camera subsystem, we had the option of keeping our current camera, the Phantom v710, or buying a much more sensitive camera that uses arrays of Avalanche Photo-Diodes (APDs). Such an optimally-sensitive camera, called the APDCAM-10G, is available commercially from a Hungarian company, Fusion Instruments, Inc.

We ultimately chose to purchase the APDCAM-10G because of its much higher sensitivity. Even though we are striving to maximize the light-collection of the optics, we are ultimately limited by the detected photon flux and thus need to maximize that. The APDCAM-10G and its corresponding microlens array are shown in Fig. 3-1.

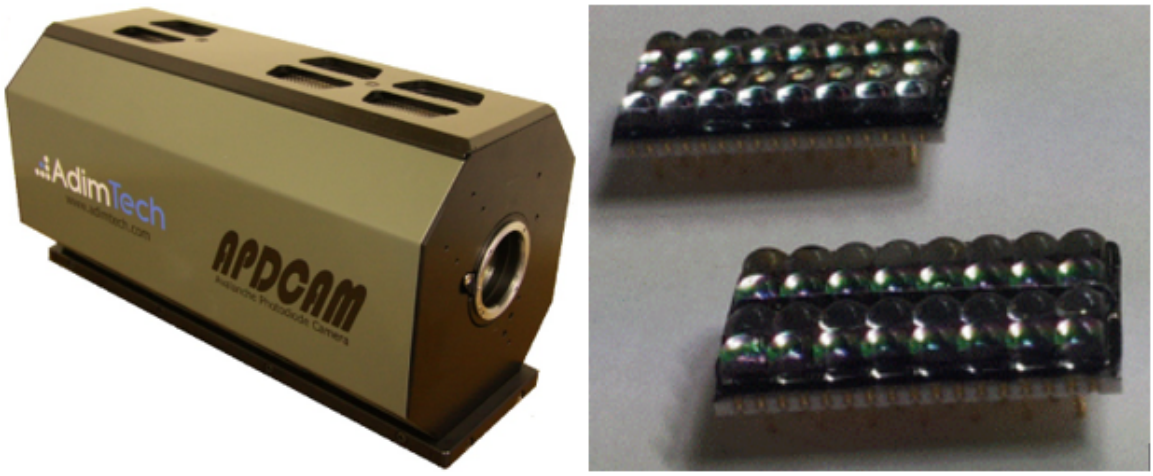

Figure 3-1: Fusion Instruments APDCAM-10G (left) and the microlens array (right). This will be the camera we will be using for the GPI diagnostic, with a 4x8 microlens array coupled to the camera for near 100% photon detection efficiency.

The two cameras and their respective optical setups, including various lenses, a vacuum window to separate the more delicate aforementioned lenses from the inside of the plasma vessel, and the turning mirror, were compared by conducting simulations using Zemax OpticStudio. This software provided insight into aspects such as light loss during transmission through the mirror and lenses before reaching the cameras, the parameters of the optical elements, the relative placement of those elements, and geometric image analyses. Because the detector areas of the two systems were different, the two systems had different requirements for the magnification between the image and the object planes. The detector size for the Phantom v710 was $\sim$4x4 mm, while the footprint for the APDCAM-10G detector was 23.4x39 mm. Therefore, to image a roughly 50x70 mm area in object space corresponding to the gas puff region, the required magnifications were $\sim$12 and $\sim$2 respectively.

In both cases, the optical distance between the object plane at the gas puff to the vacuum window was held constant at about 1.4 meters, accurately reflecting the

physical constraints by the chosen ports and their distances to the plasma. Also, the distance between the vacuum window and the camera was approximately constant since the camera had to be located outside of the port flange. However, the other optical elements were kept variable, with their separations optimized to provide the most ideal focus at the camera detection planes with appropriate magnifications. Partly as a result of these simulations, we decided on purchasing the APDCAM-10G.

3.2.1 Simulations for the Phantom v710 Optical Setup

From the plasma to the Phantom v710, the intermediate optical system included a large turning mirror, a vacuum window, a series of five focusing lenses, and an objective lens to concentrate light onto the fiber which connects to the camera. This portion of the optical setup is shown in Fig. 3-2, specifically for the 587 nm He line.

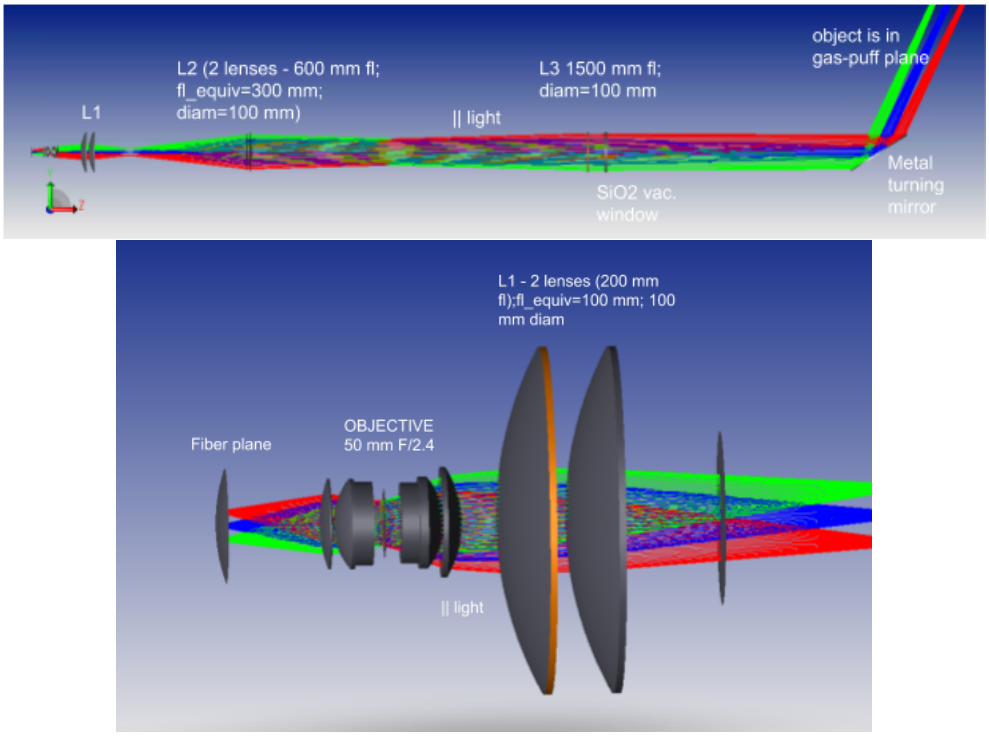

Figure 3-2: (Top) Optical system for the case where the Phantom v710 is chosen for the GPI. Photons from the plasma in the top right are reflected by the angled turning mirror, then channeled through the vacuum window, and finally through the series of lenses to the focus of the camera on the far left. (Bottom) Zoomed in view of the system of lenses and objective lens pictured in the far left of the overview.

Detailed specifications of the precise distances and other optical parameters necessary to simulate the above system are shown in Appendix Table B.1. In this case, the overall distance between the gas puff and the detection plane was approximately 2800 mm. Further details on the general characteristics of the system are available in Appendix Fig. A-2, with one of the more important characteristics being that this system had a paraxial magnification of 9.8. Although this was slightly under the desired magnification of approximately 12, the difference was slight enough that it was not significantly detrimental to the system.

Simulation results including a spot diagram to determine the effects of off-axis objects on the image focus and a geometric encircled energy plot to determine the fraction of enclosed photon energy based on radius on the image plane are shown in Appendix Figs. A-3 and A-4. As seen from the image spot diagrams for single points in the object plane, off-axis objects caused aberrations to the image, and the image ended up being on the scale of 0.4 to 1.4 mm in size. The geometric encircled energy plot shows over 90% of the photon energies being within an image spot of 0.35 mm radius.

However, while the system seemed to be acceptable, upon closer inspection, it turned out that photons were lost in the system, particularly in the objective lens, due to the limiting geometric factors not allowing all of the photons at the edge of the inner lenses to pass through. Due to the loss of photons in transit from the plasma boundary to the detection plane, we sought for a better option.

3.2.2 Simulations for the APDCAM-10G Optical Setup

From the plasma to the APDCAM-10G, the intermediate optical system consisted of a large turning mirror, a vacuum window, and only three focusing lenses to direct light onto the detector of the camera. The small magnification, being approximately 2, caused this system to be much less complicated than that for the Phantom v710. The overview of this optical setup is shown in Fig. 3-3, specifically for the 587 nm He line.

Detailed specifications of the precise distances and other optical parameters neces-

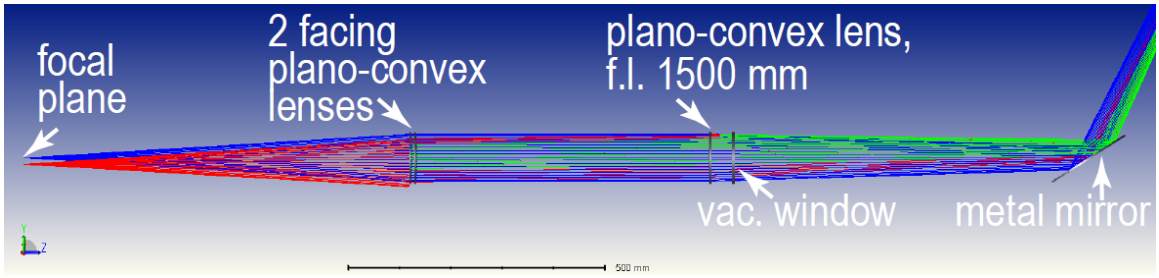

Figure 3-3: Optical system for the case where the APDCAM-10G is chosen for the GPI. Photons from the plasma in the top right are reflected by the angled turning mirror, then channeled through the vacuum window, and through three lenses to focus onto the camera.

sary to simulate the above system are shown in Appendix Table B.2. In this case, the overall distance between the gas puff and the detection plane was also approximately 2800 mm as required. Further details on the general characteristics of the system are available in Appendix Fig. A-5, with one of the more important characteristics being that this system had a paraxial magnification of approximately 2, as desired.

Similar to the simulations run for the Phantom v710, an image spot diagram and a geometric encircled energy plot for points in object space were generated, and they are shown in Appendix Figs. A-6 and A-7. From the spot diagram, off-axis objects caused slight aberrations to the image as well, but the image still ended up being about 0.1 to 0.2 mm in size, well within the size constraints of the detector. The geometric encircled energy plot further supported this, with over 90% of the photon energies being within 0.06 mm of the radius of the image spot.

Similar to the Phantom v710 system, this configuration also faced some loss of photons during transmission. However, because of the simpler design with a fewer number of lenses, the magnitude of photons lost was not as significant. Moreover, because of the excellent focus onto the detector and the near-perfect efficiency of the detector, the APDCAM-10G system outperformed the Phantom v710 configuration to become our camera of choice.

3.3 Gas-Puff Valve Subsystem

For the gas-puffing subsystem, we needed a system that would contain and store the gas until a puff is to be released into the plasma vessel in a fast manner, while also having the capability of keeping track of absolute and differential pressures within the system to determine how much gas is released into the vessel per puff. A CAD rendering of all of the components for this subsystem is shown in Fig. 3-4.

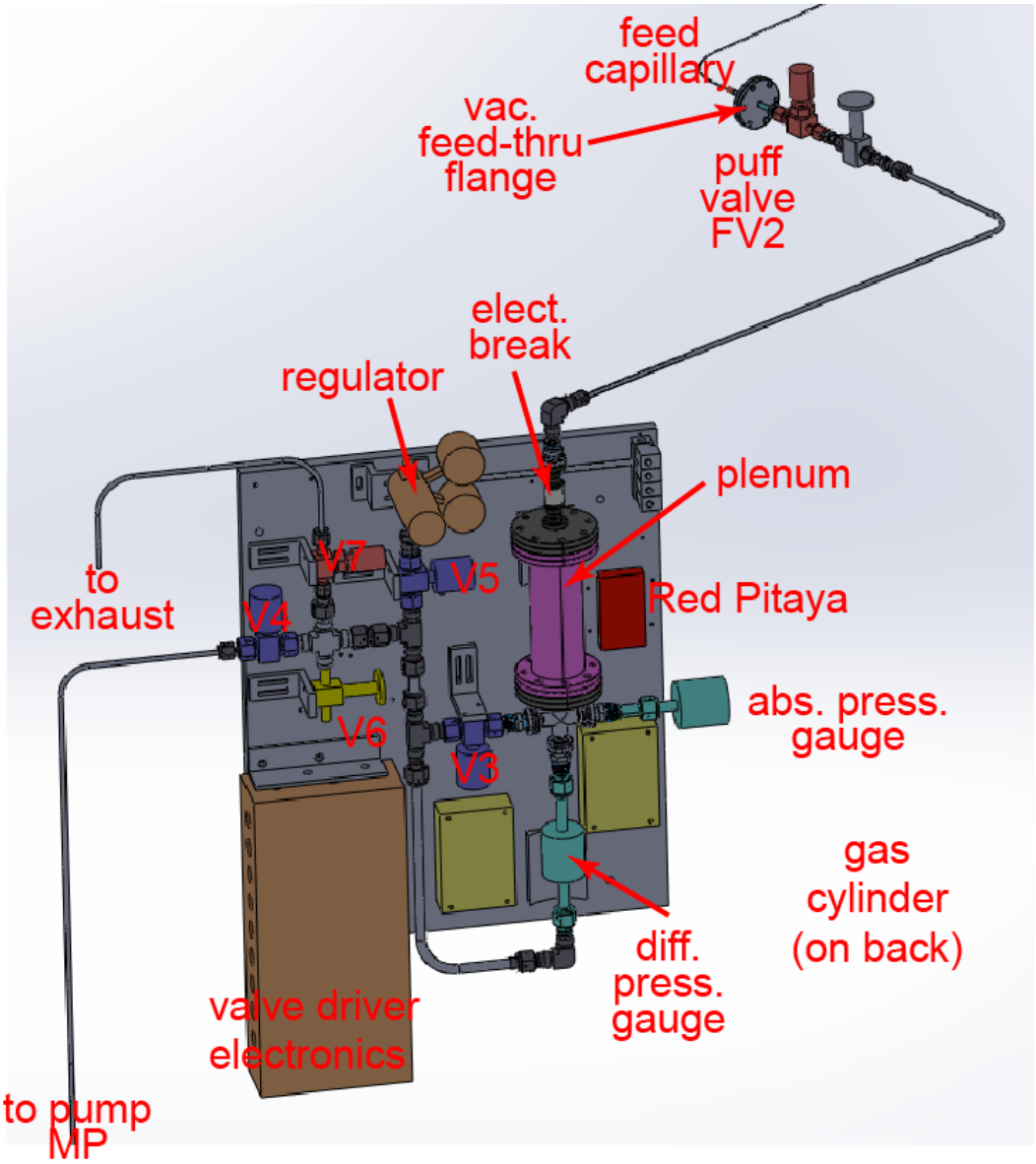

Figure 3-4: CAD rendering of the gas-puff valve subsystem with all of the components in place on the mounting plate.

As seen in the rendering, there are multiple components that make up the subsystem as a whole.

(1) To store the gas, there is a replaceable gas cylinder on the back of the mounting plate (not visible in Fig. 3-4) that provides gas for the system.

(2) To control the flow of the gas, a series of bellows-sealed pneumatically actuated valves are utilized. These valves were purchased from Swagelok—FV2 is the "injection" valve that opens and closes quickly to release gas into the feed capillary, which goes to the nozzle and into the plasma vessel, while V3, V4, and V5 are "slow" valves which facilitate the transfer of gas within the main valve system to the plenum or to the vacuum pump. The plenum serves as the reservoir for the gas being injected and provides backing pressure for the injection.

(3) To keep track of how much gas is in the valve system and how much is released into the plasma vessel, an absolute pressure gauge and a differential pressure gauge from MKS were purchased, since the amount of gas held in the system correlates directly to the pressure within.

(4) To be able to control the Swagelok valves electronically, we purchased pneumatic solenoid valves from Numatics, which serve as links between electrical commands and physically actuating the Swagelok valves to open and close. These valves are mounted on the top right of the plate and connect to the Swagelok valves through tubes that allow air to be forced from the solenoid valves to actuate the Swagelok valves.

(5) To be able to control the Numatics valves remotely, we utilize a Red Pitaya field-programmable gate array (FPGA). The Red Pitaya is also a computer and is therefore on the network and in principle can be accessed from any other computer on the network. The Red Pitaya also digitizes the data from the pressure gauges, therefore allowing measurement and archival of how much gas is released to the vessel.

(6) Because the input/output pins on the Red Pitaya have set voltage limitations that are insufficient for the solenoid valves, an intermediate system through the valve driver electronics is needed. In those control circuits, the 3.3 V signals from the Red Pitaya are taken and amplified to the 24 V needed for solenoid valve control. The control circuit for the fast valve also includes inputs for timing triggers and permission from W7-X to release gas into the vessel.

(7) For safety purposes, there is a regulator and an electrical break.

Much of the initial design and refinement of the system was done by James L. Terry, though I was able to help in choosing to purchase the Numatics solenoid valves and the parts needed to connect them to the Swagelok valves. I also aided in physically assembling the main part of the subsystem on the bench, including designing the mounting brackets and holding structures.

3.4 Valve Control and Data Collection Subsystem

For the valve control and data collection subsystem, we opted to create a Python graphical user interface (GUI) coupled with programming a Red Pitaya STEMlab 125-14 FPGA board. As seen in the previous section, the Red Pitaya is to be mounted alongside the gas-puff valve subsystem and connected to the valve driver electronics to be able to drive the solenoid valves. As noted above, the Red Pitaya also takes in readings from the absolute and differential pressure gauges as well as inputs from W7-X about timing triggers and firing permissions.

3.4.1 Red Pitaya Field-Programmable Gate Array

Programming the Red Pitaya FPGA entailed sending the Red Pitaya a blueprint of how to rearrange its structure and what signals to respond to, as it is an integrated circuit designed to be configured by the consumer on the field. With the guidance of PhD Student William McCarthy, I was able to take charge in the design of the Red Pitaya. A schematic overview of the Red Pitaya is shown in Fig. 3-5.

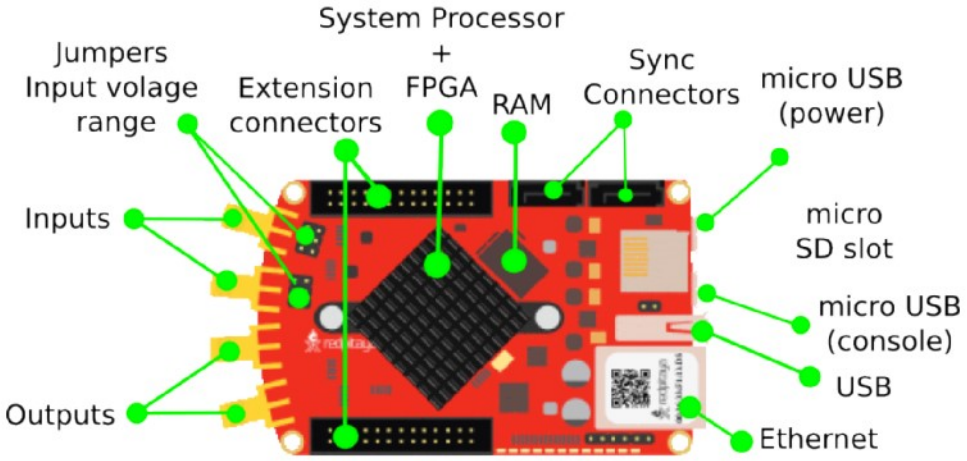

Figure 3-5: General schematic of the Red Pitaya STEMlab 125-14 components. The Red Pitaya has on-board processor and FPGA, along with RAM and network connectivity on top of customizable pins in the extension connectors [3].

The Red Pitaya features two fast-digitizing analog inputs and two analog outputs. Furthermore, the Red Pitaya also contains four slow analog inputs and four outputs, but those pins were not available during FPGA programming. As such, we utilized the remaining 16 single ended digital input/output pins. The schematic of the Red Pitaya extension connector pinout is shown in Fig.3-6.

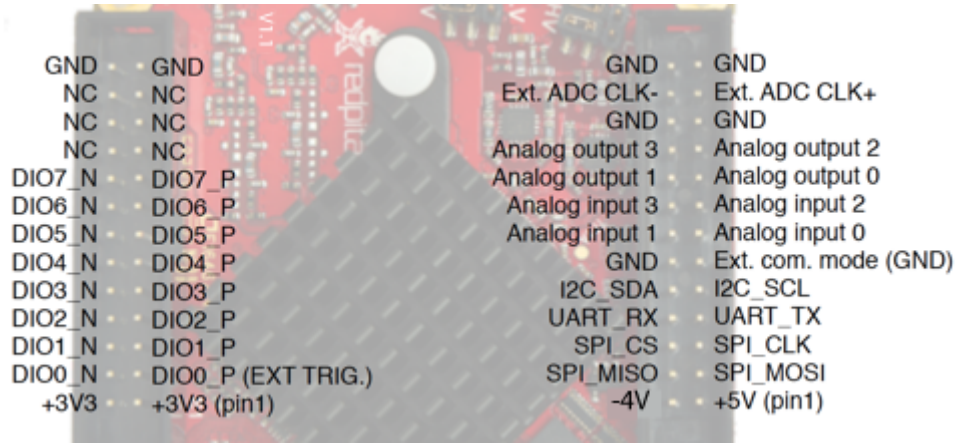

Figure 3-6: Detailed pin descriptions of the extension connectors [4].

To program the FPGA, I used the Vivado Design Suite in conjunction with code

to set up the pins, components, and variables needed, and then connected appropriate pins to locations within blocks and vice versa, with the general connections as shown in Fig. 3-7. The Vivado detailed result is shown in Appendix Fig. A-8.

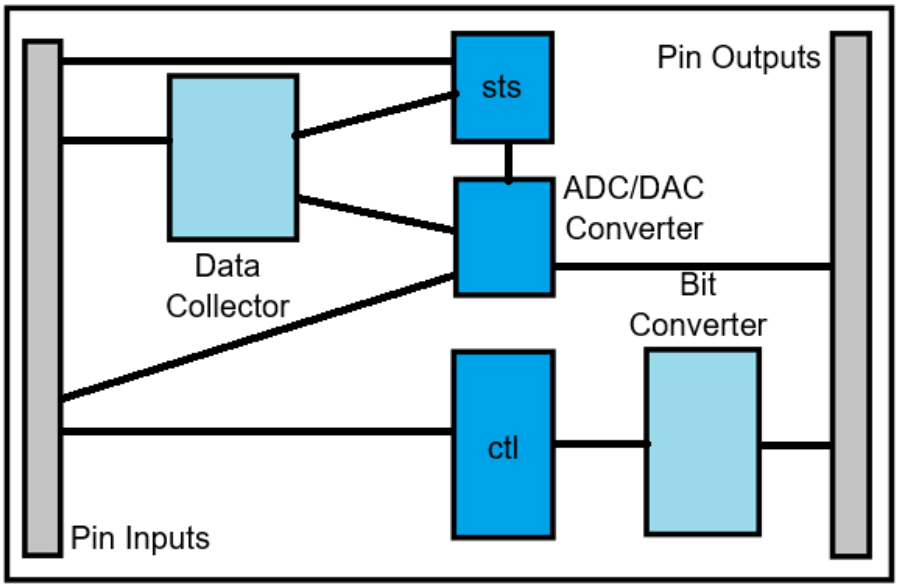

Figure 3-7: General overview of the FPGA design layout.

The pins are represented by the gray areas on the far left and far right, while the inner blocks represent specific functions taken between pins. Notably, the sts block represents the status register, which accepts signals to set internal states within the FPGA. The ctl block represents the control register, which passes signals outward to the pins, thereby controlling them. However, because the signals sent by the control register are 32 bits, while only 1 bit is needed to control the output pins, an intermediate bit converter block is needed to truncate the signal for the pins.

Various pieces of code were needed to generate the design in Vivado Design Suite and served different purposes in programming the FPGA pins to the appropriate variables and to set functions available to us from the software end. The code is shown in Appendix C, with the functions as follows:

- block_design.tcl (Appendix C-1): This is the overall block design file that allows Vivado Design Suite to generate the graphical version shown in Appendix Fig. A-8. All additions of blocks, pins, and connections are done in this file.

- config.yml (Appendix C-2): This is the configuration file for Vivado Design Suite, which provides settings such as the cores/blocks to be used, the control and status registers, and other parameters.

- extension_connector.xdc (Appendix C-3): This defines the digital I/O pins from the Red Pitaya and maps them to corresponding internal variables.

- GPI_2.hpp (Appendix C-4): This is a C++ header file that creates all of the functions to read and write to the status and control registers through the variable names set in the block design and the extension connector files. This file also defines the functions needed for data collection.

- GPI_2.py (Appendix C-5): This is a Python file that links to the previous header file, mapping Python functions to the functions within the header file. With this, we can control the FPGA using Python scripts. The data collection functions were not implemented in this file at the time.

- DataCollect.vhd (Appendix C-6): This sets up the data collection block. The original file was provided by William McCarthy for his "Mirror Langmuir Probe" project and the file has been simplified since.

- outputs.vhd (Appendix C-7): This sets up the intermediate block to process the signal from the control register to the outbound pins.

With all of these files in place, the FPGA design was uploaded to the Red Pitaya successfully. Subsequent work with the design was picked up by Sean Ballinger to complete the implementation of the data collection functions.

3.4.2 Python Graphical User Interface

To control the Red Pitaya's functions from a computer, I created a GUI in Python using the 'Tkinter' library. The GUI is shown in Fig. 3-8.

The Python GUI shows a stripped down overview of the valve system as well as specific statuses of the valves and W7-X. With this GUI, it is possible to control each

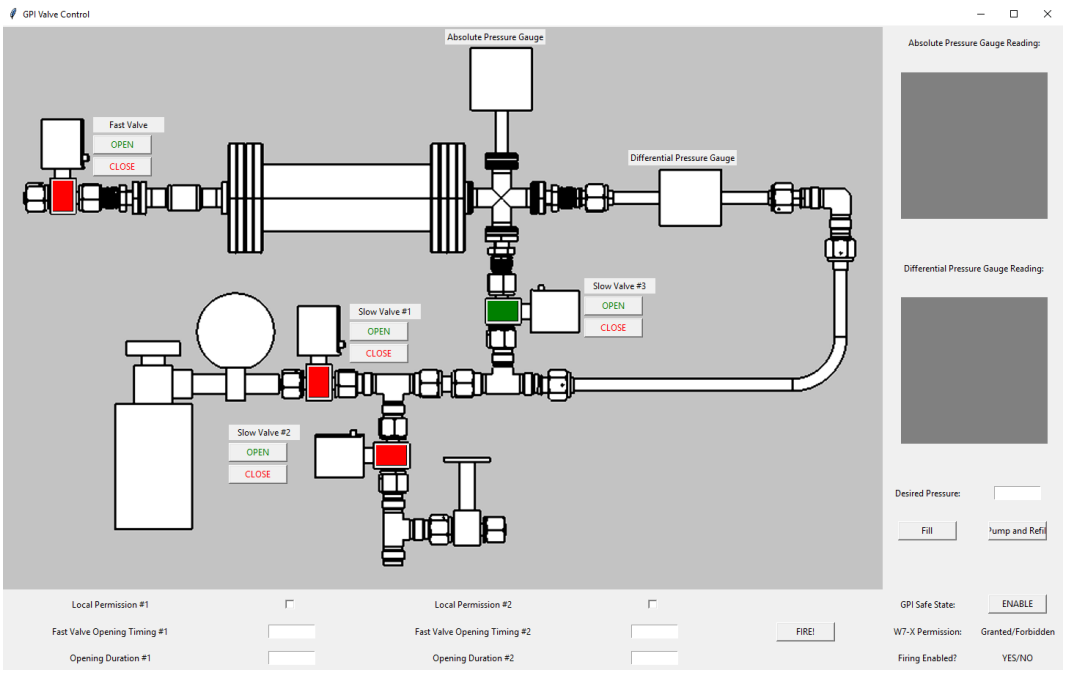

Figure 3-8: Overview of the Python GUI.

of the four valves manually, opening and closing them individually. However, it is also possible to set automatic timings for the valves to open and close, releasing gas into the plasma vessel at set instances and for set durations after receiving permission and timing triggers from W7-X. The boxes on the right are for readings from the differential and absolute pressure gauges, which measure how much gas is stored in the system and how much gas is injected through the "injection" valve, FV2. The GUI coupled with the FPGA allows the system to automatically fill or refill to a user inputted desired pressure, and stores relevant data to the server.

The GUI is able to communicate with the Red Pitaya through a network connection and then calls upon the Python functions written for the FPGA control as mentioned in the previous section. Thus, the various abilities seen on the front-end of the GUI are mapped to functions in the Red Pitaya through the back-end. The code to generate the GUI is shown in Appendix C-8.

Subsequent work was taken up by Sean Ballinger, completing the graphing mechanisms for readings from the pressure gauges and the automatic filling functionality.

3.5 Heat Loading Simulations

In order to determine the heat loads experienced by the plasma-facing components of the diagnostic, which would, in turn, determine the design for cooling mechanisms, heat loading simulations were conducted using the COMSOL Multiphysics code. This is a finite element code. Because of W7-X's long-term goal of attaining steady-state operation for 30 minutes, it is imperative that the diagnostic's components will be kept within safe operating limits. The simulations included steady-state runs for the gas puffing nozzle at AEK50 and for the overall optical system within AEE50. All of the modeling assumed a 100 kW/m^2 radiative heat load emanating from the plasma. The heat load is the maximum expected during the steady-state operation phase of W7-X. In the COMSOL thermal modeling, the heat load was simulated as a blackbody sheet radiating at 880°C roughly 200 mm beyond the port entrances.

3.5.1 Heat Load Analysis of the Gas Puff Nozzle

With the gas puffing nozzle as designed by James L. Terry, running the steady-state heat load simulation results in the temperature distribution shown in Fig. 3-9.

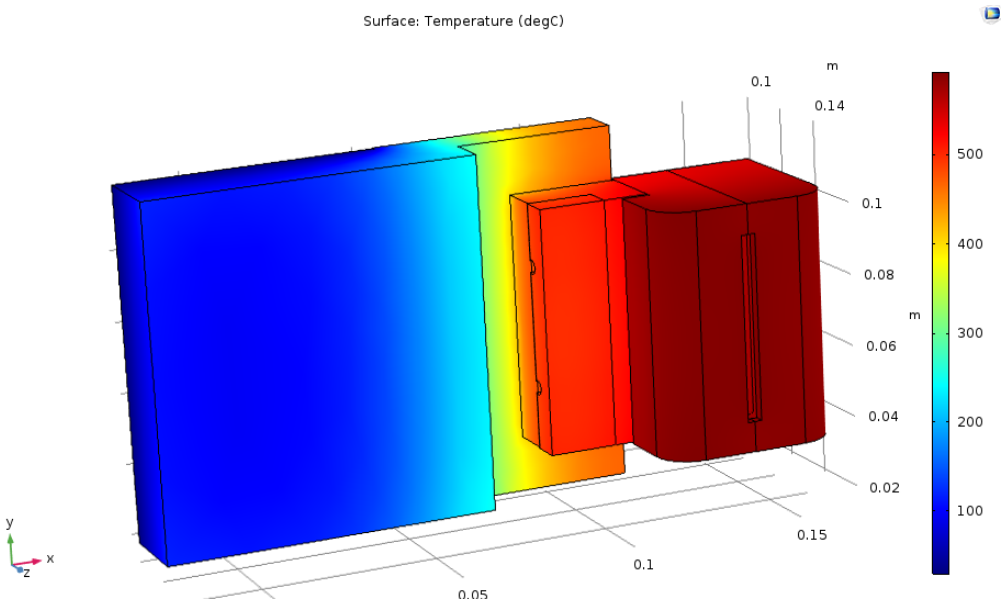

Figure 3-9: COMSOL analysis of the gas puff nozzle with the back panel cooled.

This steady-state simulation was run with the boundary condition that the bracket on which the nozzle structure is fastened is in contact with a surface being held at 30°C, as indicated by the dark blue region on the left in Fig. 3-9. The nozzle's material was graphite, while everything else was modeled as stainless steel. With those thermal properties, the gas puff nozzle reaches a maximum of 590°C at its plasma-facing surface.

In an effort to reduce the peak temperature of the gas puffing nozzle, another COMSOL simulation was run with the nozzle design as shown in Fig. 3-10.

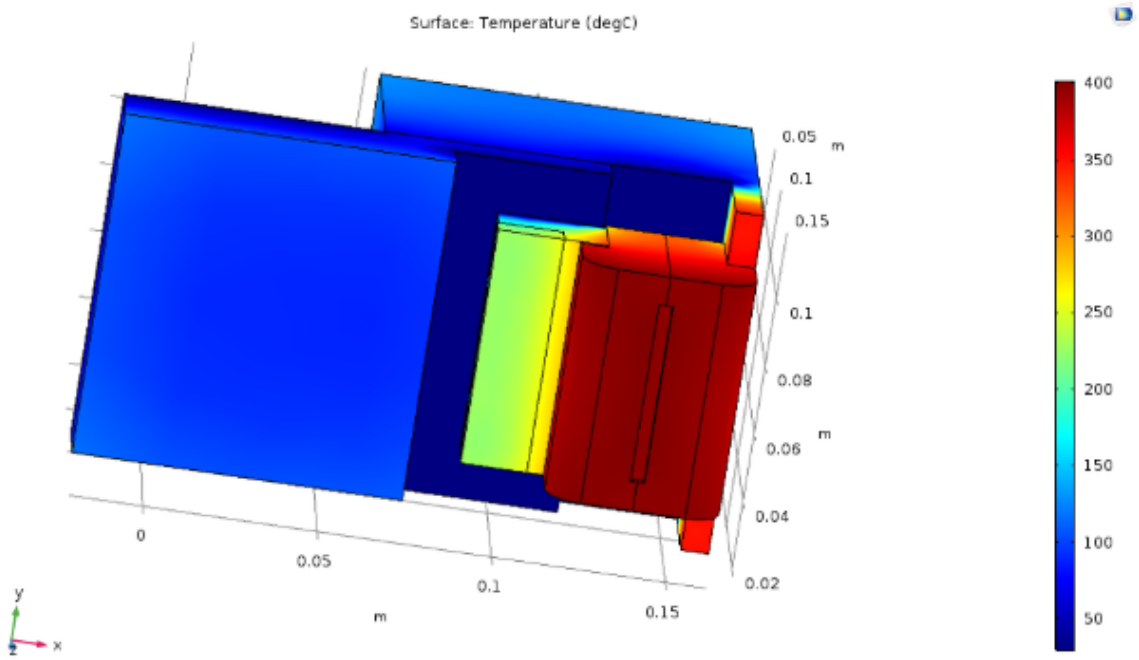

Figure 3-10: COMSOL analysis of an idealized situation for the gas puff nozzle with a larger fraction of the mounting bracket cooled to 30°C.

Material conditions were the same as the previous simulation, with the nozzle component made from graphite and everything else being stainless steel. With additional cooling at 30°C provided to the nozzle (to the right of the nozzle as seen in Fig. 3-10) through the mounting bracket, the maximum temperature reached on the face of the nozzle is 400°C. Because this situation is not realistic, as it would be very difficult to provide appropriate cooling to the nozzle on that side which is not directly attached to the port wall, this constitutes the best-case scenario.

With the peak temperature of the nozzle being about 600°C, even though that is not the best case scenario, the W7-X engineers believe that this should not be a danger to machine operation, nor should it damage the nozzle structure.

3.5.2 Heat Load Analysis of the Immersion Tube

Port AEE50 hosts the immersion tube which constitutes the vacuum interface and houses the optical system. Due to the sensitive elements within the component, it is imperative that temperatures do not reach high enough to compromise the vacuum window (whose maximum tolerance for temperature is 200°C), or affect the performance of the lenses or turning mirror. A steady-state simulation including water-cooling pipes to the tube portion beyond the vacuum window (but not extending to the shutter/turning mirror box at the plasma end of the immersion tube) resulted in Fig. 3-11.

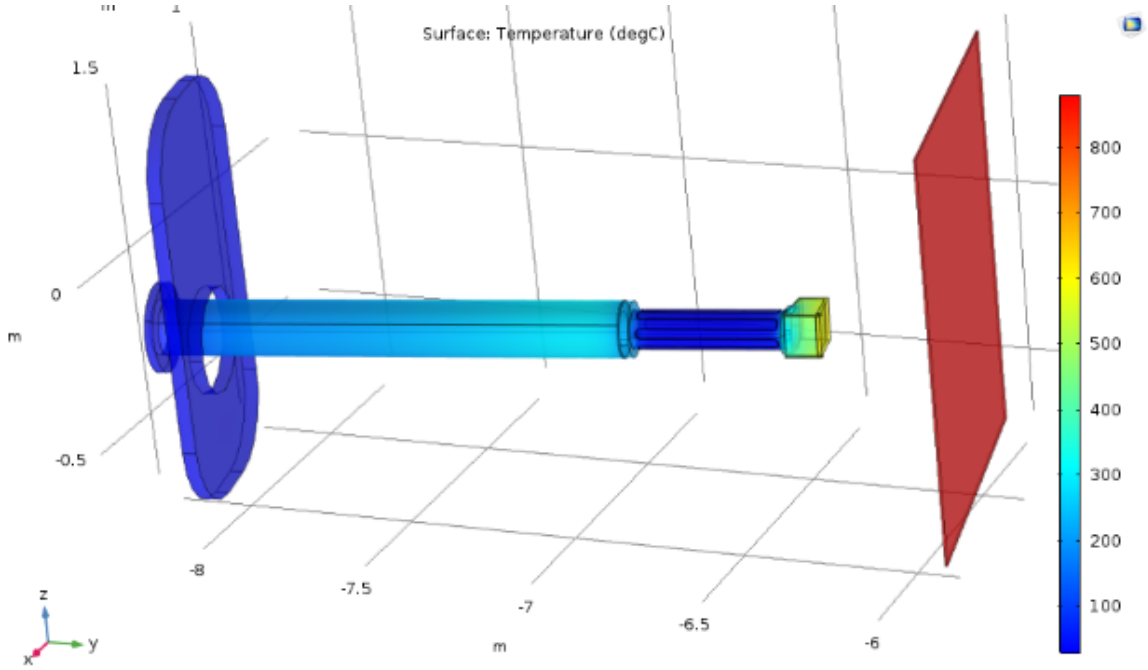

Figure 3-11: COMSOL analysis of the optical system with one set of cooling pipes. Port AEE50 can be seen on the left, while the 880°C plasma sheet can be seen on the right.

The cooling pipes in thermal contact with the tube were held at 30°C, like the

cooling for the gas puff nozzle. All components in this simulation were modeled after stainless steel. In this configuration, the front face of the shutter box, which also hosts the turning mirror, reaches about 570°C. The vacuum window connecting the bigger tube to the smaller tube reaches about 230°C. While the shutter box can withstand temperatures of that scale, the vacuum window should be below 200°C to be within safety margins.

As such, a subsequent steady-state simulation with a second set of cooling pipes added resulted in Fig. 3-12.

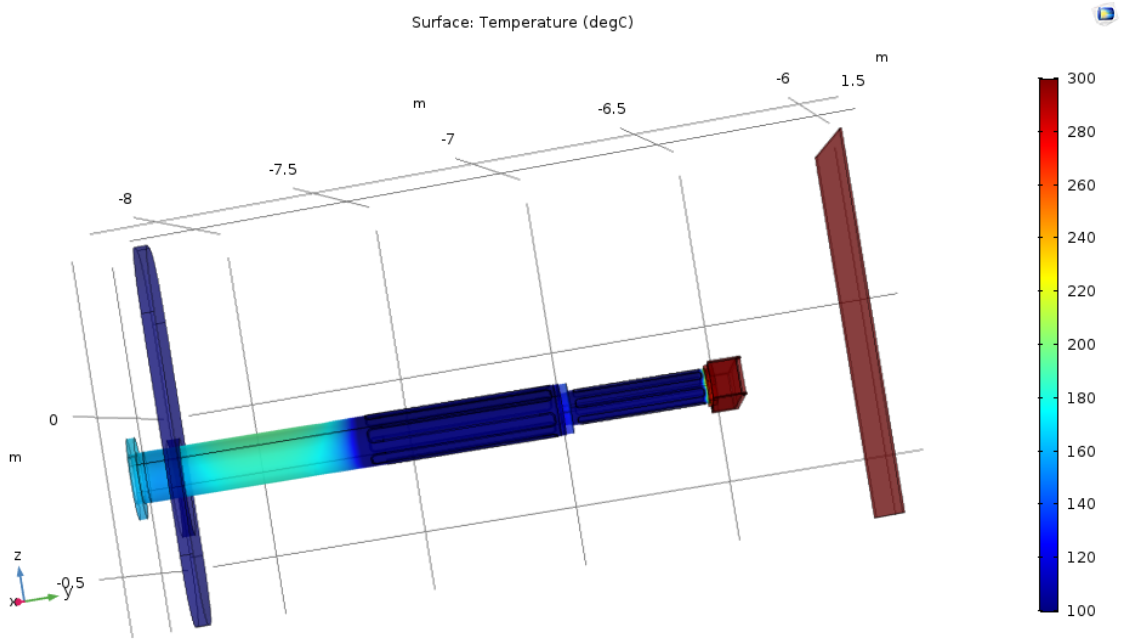

Figure 3-12: COMSOL analysis of the optical system with two sets of cooling pipes flanking the vacuum window to cool the length of the tube as much as possible and therefore the window as well.

With the two sets of cooling pipes, the shutter box still reaches about 570°C but the vacuum window is about 175°C, now within the safety margins. Thus, we found that two sets of cooling pipes are necessary to keep the window, the tubes, and optical elements sufficiently cooled.

Chapter 4

Summary

At present most of the individual components of the GPI diagnostic have been designed. CAD renderings of the system as presently designed and mounted onto W7-X are shown in Figs. 4-1 and 4-2. In Fig. 4-1 the roles of each of the subsystems are labeled.

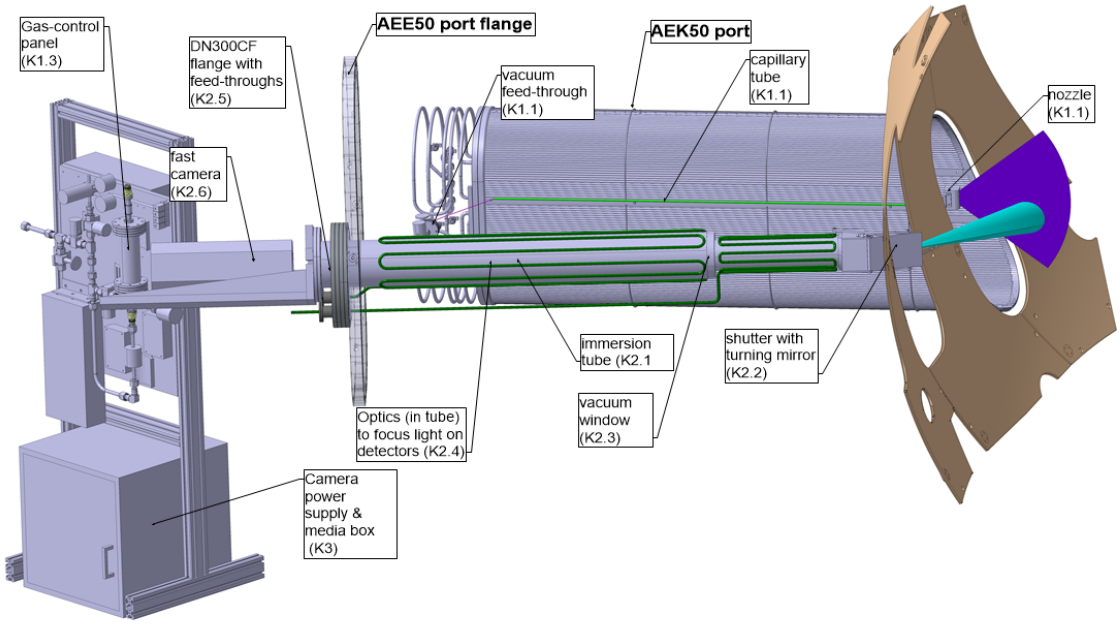

Figure 4-1: Overview of all of the components of the GPI diagnostic in place at their respective port locations.

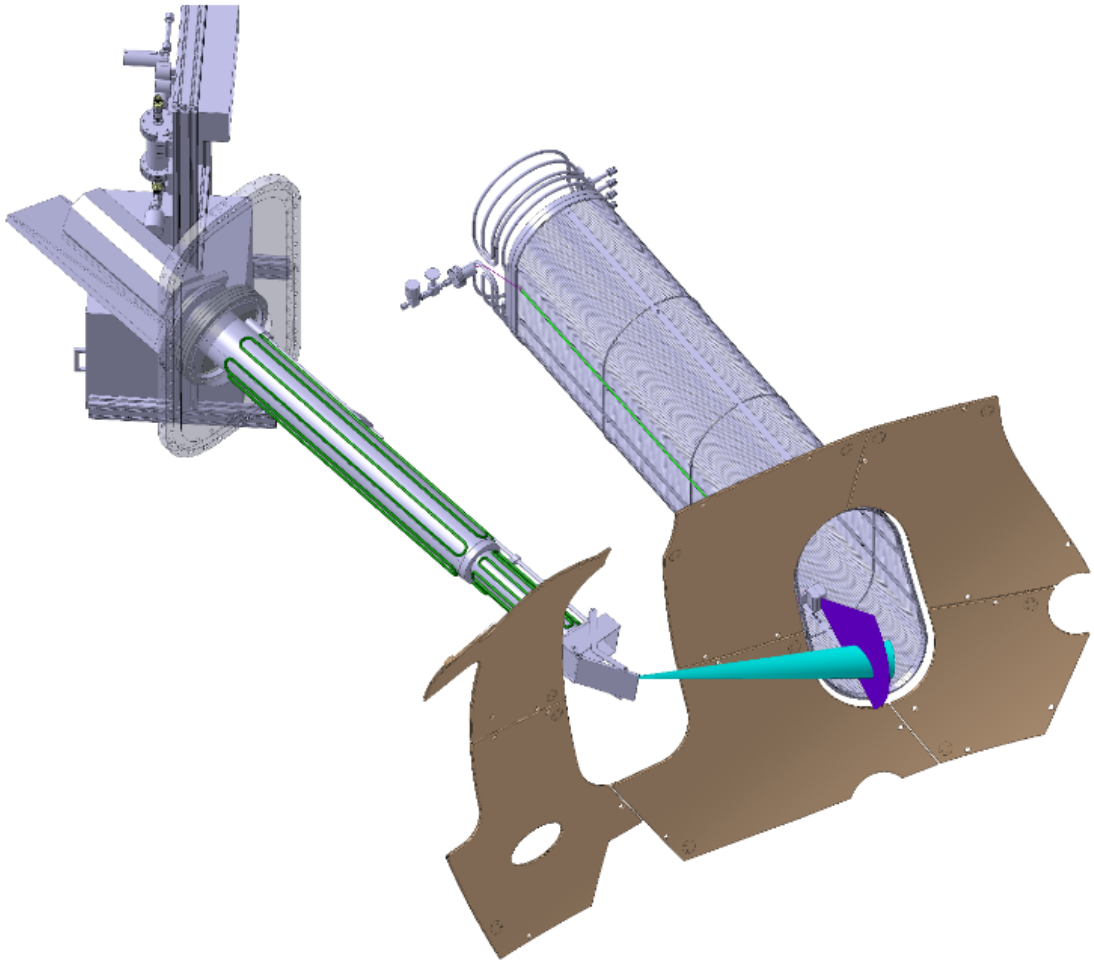

Figure 4-2: Overview of the entire GPI diagnostic from a different angle than in Fig. 4-1. The sightline intersecting with the injected gas can be clearly seen.

Within the overall design of the entire diagnostic and the past two years of working with the group, I now summarize my main roles in the project:

1. On-site at W7-X in Germany, I helped install the fast-framing camera diagnostic. I performed the alignment of the camera system with the image-guide.

2. I helped narrow down the selection of ports in the initial search for suitable ports on which to mount the GPI diagnostic.

3. I performed simulations of the optical system using Zemax OpticStudio for both the Phantom v710 and APDCAM-10G cases.

4. I helped assemble the gas-puff valve subsystem.

5. I was responsible for programming the Red Pitaya FPGA and its corresponding Python GUI.

6. I performed finite element code simulations of the expected heat loads on the gas puffing nozzle and the immersion tube that houses the optical system and the vacuum window. The code used was COMSOL Multiphysics.

The fast-framing camera has run routinely and successfully on W7-X since its installation. The GPI diagnostic has passed W7-X's Conceptual Design Review. Although it must still pass a Final Design Review and have the remaining components fabricated or procured, it is on track to be installed onto W7-X by the beginning of its second operation phase in 2021.

Appendix A

Figures

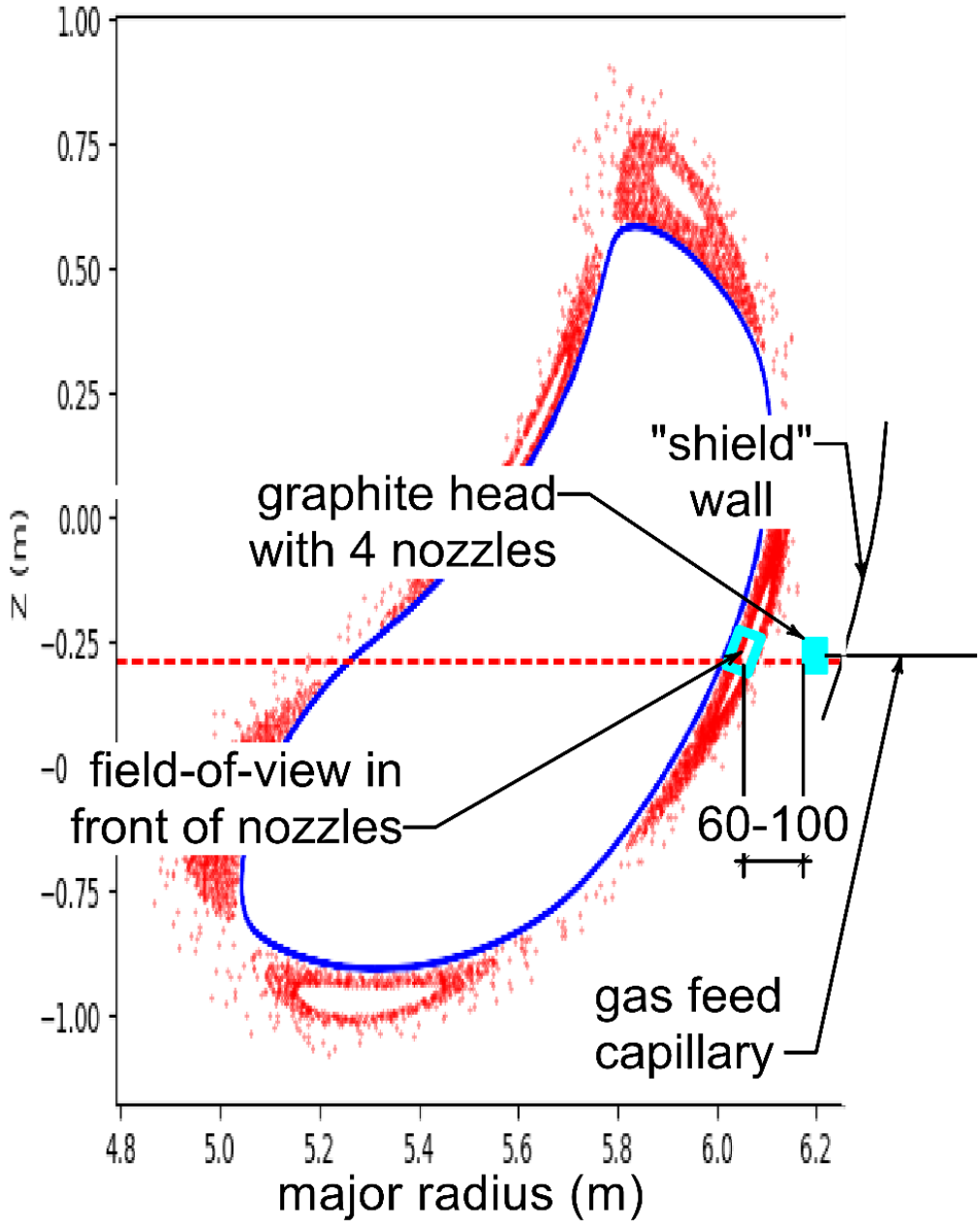

Figure A-1: Plasma cross section at the toroidal angle of the gas puff nozzle at port AEK50. Prominent plasma features such as the LCFS in blue and plasma islands in red are also shown.

```
GENERAL LENS DATA:

Surfaces                 :                 25
Stop                     :                  6
System Aperture          :   Object Cone Angle =     12
Fast Semi-Diameters      :   On
Field Unpolarized        :   On
Convert thin film phase to ray equivalent : On
J/E Conversion Method    :   X Axis Reference
Glass Catalogs           :   SCHOTT MISC INFRARED HERAEUS
Ray Aiming               :   Off
Apodization              :   Uniform, factor =     0.00000E-
Reference OPD            :   Exit Pupil
Paraxial Rays Setting    :   Ignore Coordinate Breaks
Method to Compute F/#    :   Pupil Size/Position
Print Coordinate Breaks  :   On
Multi-Threading          :   On
OPD Modulo 2 Pi          :   Off
Temperature (C)          :      2.00000E+001
Pressure (ATM)           :      1.00000E+000
Adjust Index Data To Environment  : Off
Effective Focal Length   :          -154.6457    (in air at
Effective Focal Length   :          -154.6457    (in image s
Back Focal Length        :           2232.404
Total Track              :           2017.432
Image Space F/#          :           6.318865
Paraxial Working F/#     :           23.06932
Working F/#              :           22.96051
Image Space NA           :         0.02166872
Object Space NA          :          0.2079117
Stop Radius              :           6.755809
Paraxial Image Height    :           41.18969
Paraxial Magnification   :           9.807069
Entrance Pupil Diameter  :           24.47365
Entrance Pupil Position  :           27.56974
Exit Pupil Diameter      :           90.54231
Exit Pupil Position      :           2031.232
Field Type               :   Object height in Millimeters
Maximum Radial Field     :                4.2
Primary Wavelength [µm]  :             0.5876
Angular Magnification    :          0.2703014
```

Figure A-2: Various details for general characteristics of the Zemax simulated Phantom v710 optical system.

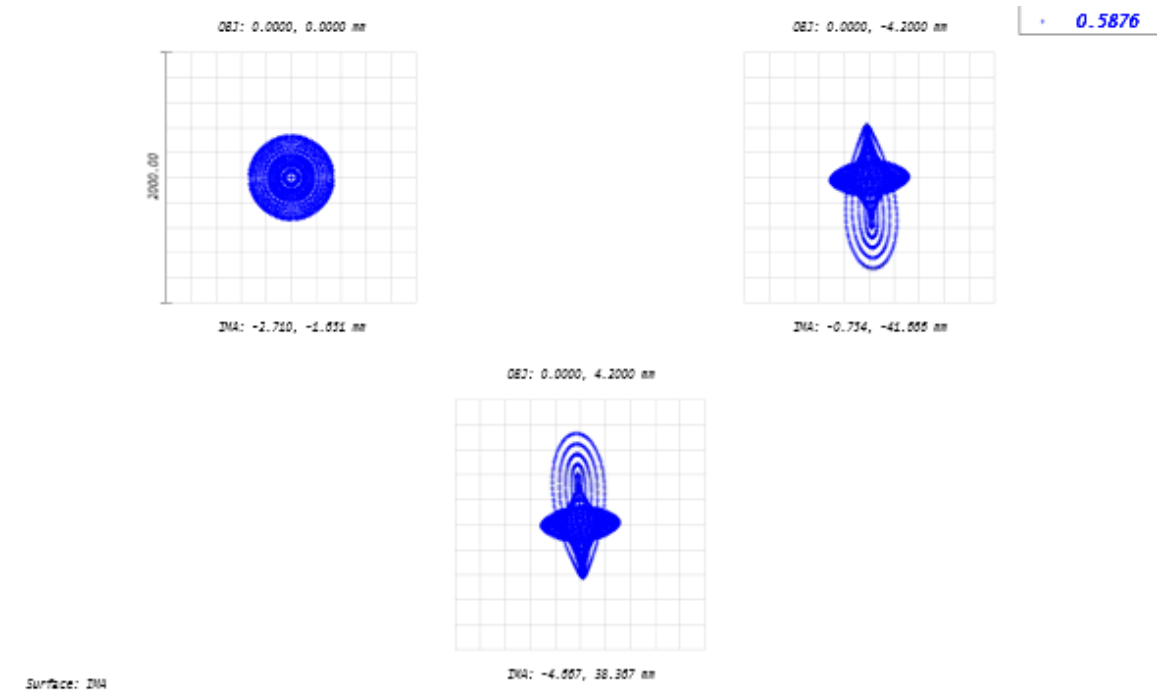

Figure A-3: Spot diagram for the Phantom v710 simulation. The image on the top left is that of the on-axis object, whereas the other two correspond to off-axis objects. We can thus see that the focused image spot is at best 0.4 mm in size and at worst 1.4 mm, which is acceptable for the 4x4 mm detector.

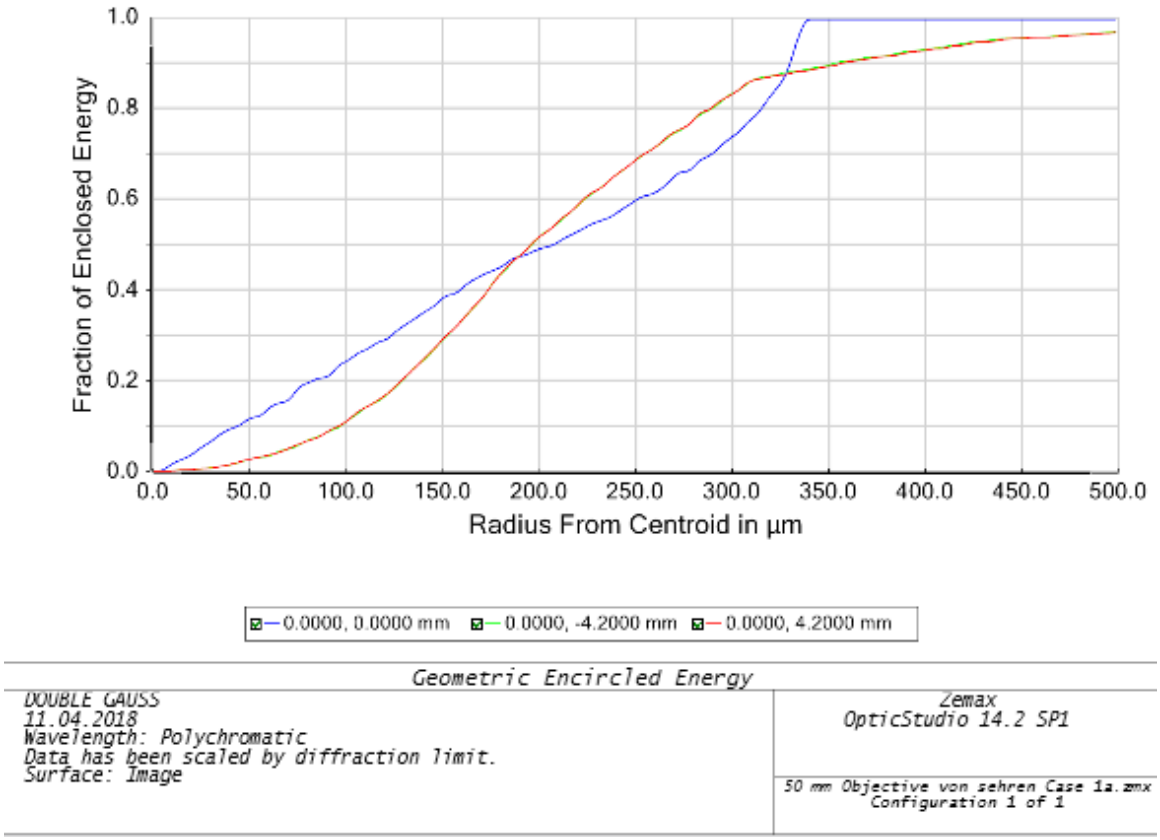

Figure A-4: Geometric encircled energy plot for the Phantom v710 simulation. The on-axis case reaches 100% enclosed photon energy at around 0.34 mm, while the off-axis cases reach about 90% at that point.

```
GENERAL LENS DATA:

Surfaces                  :                    11
Stop                      :                     7
System Aperture           :       Float By Stop Size =    45
Fast Semi-Diameters       :       On
Field Unpolarized         :       On
Convert thin film phase to ray equivalent :     On
J/E Conversion Method     :       X Axis Reference
Glass Catalogs            :       SCHOTT MISC INFRARED HERAEUS
Ray Aiming                :       Off
Apodization               :       Uniform, factor =          0.0
Reference OPD             :       Exit Pupil
Paraxial Rays Setting     :       Ignore Coordinate Breaks
Method to Compute F/#     :       Tracing Rays
Method to Compute Huygens Integral   : Force Spherical
Print Coordinate Breaks   :       On
Multi-Threading           :       On
OPD Modulo 2 Pi           :       Off
Temperature (C)           :           2.00000E+01
Pressure (ATM)            :           1.00000E+00
Adjust Index Data To Environment  :     Off
Effective Focal Length    :               670.5605        (in ai
Effective Focal Length    :               670.5605        (in im
Back Focal Length         :                577.533
Total Track               :               1323.011
Image Space F/#           :               1.500551
Paraxial Working F/#      :               16.00376
Working F/#               :               15.97158
Image Space NA            :             0.03122742
Object Space NA           :             0.06070368
Stop Radius               :                     45
Paraxial Image Height     :               45.08723
Paraxial Magnification    :              -1.946564
Entrance Pupil Diameter   :               446.8761
Entrance Pupil Position   :               2923.243
Exit Pupil Diameter       :               90.00022
Exit Pupil Position       :               1430.342
Field Type                :       Object height in Millimeters
Maximum Radial Field      :               23.16247
Primary Wavelength [µm]   :                 0.5876
Angular Magnification     :               -4.96529
```

Figure A-5: Various details for general characteristics of the Zemax simulated APDCAM-10G optical system.

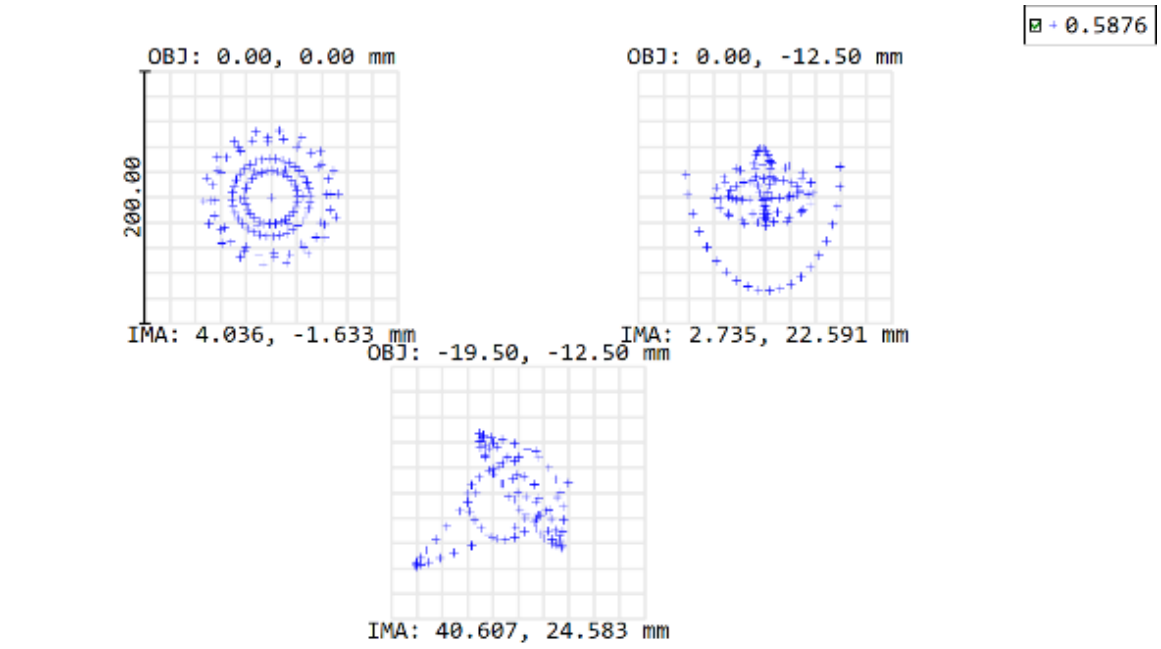

Figure A-6: Spot diagram for the APDCAM-10G simulation. The image on the top left is that of the on-axis object, whereas the other two correspond to off-axis objects. We can thus see that the focused image spot is at best 0.1 mm in size and at worst 0.2 mm, which is well acceptable for the 23.4x39 mm detector.

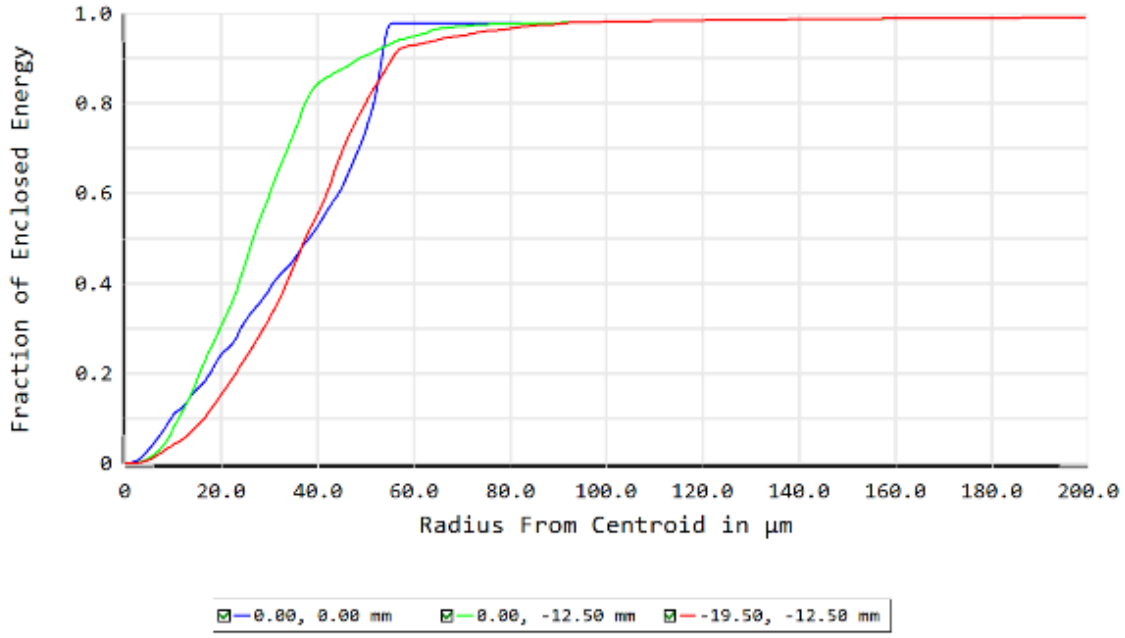

Figure A-7: Geometric encircled energy plot for the APDCAM-10G simulation. The on-axis case reaches 100% enclosed photon energy at around 0.055 mm, while the off-axis cases reach about 90% at that point.

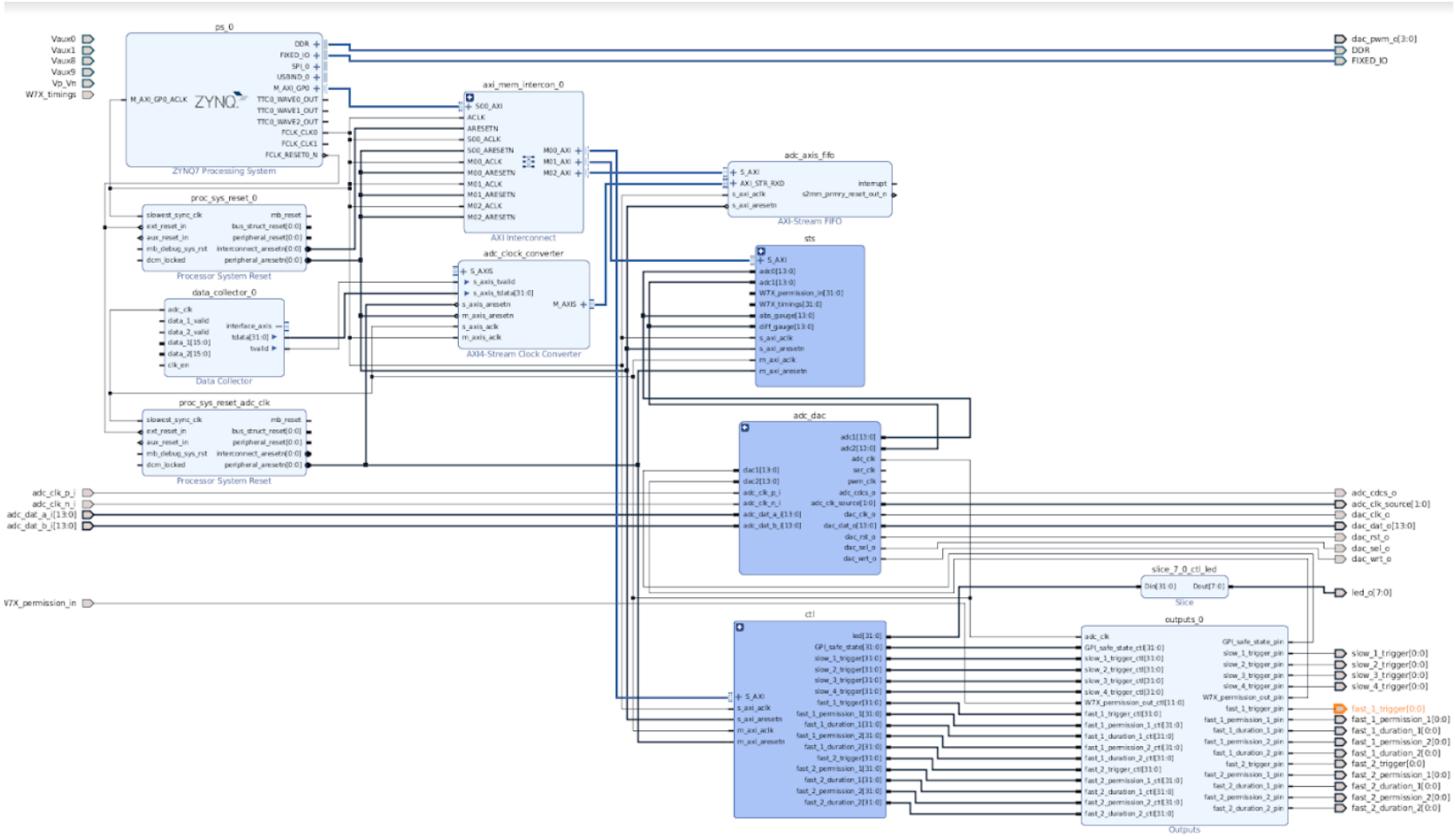

Figure A-8: Overview of the FPGA design in Vivado Design Suite.

Appendix B

Tables

	Surf:Type		Comment	Radius	Thickness		Material	Coating	Semi-Diameter	
0	OBJECT (aper)	Standard		50.000	30.000				4.200	
1	(aper)	Standard		36.600	3.731		SK16	AR	13.000	U
2	(aper)	Standard		-107.100	0.272			AR	13.000	U
3	(aper)	Standard		20.117	5.893		SK16	AR	13.000	U
4	(aper)	Standard		Infinity	2.054		F5		12.000	U
5	(aper)	Standard		13.976	6.761			AR	9.000	U
6	STOP	Standard		Infinity	7.753				10.000	U
7	(aper)	Standard		-12.115	2.054		F5	AR	8.111	U
8	(aper)	Standard		Infinity	7.616		SK16		11.600	U
9	(aper)	Standard		-19.557	0.272			AR	13.200	U
10	(aper)	Standard		-82.970	4.758		SK2	AR	15.300	U
11	(aper)	Standard		-29.460	53.922			AR	15.900	U
12	(aper)	Standard		103.000	15.000		N-BK7	AR	50.000	U
13	(aper)	Standard		Infinity	5.000			AR	50.000	U
14	(aper)	Standard		103.000	15.000		N-BK7	AR	50.000	U
15		Standard		Infinity	383.900				17.396	
16	(aper)	Standard		Infinity	5.000		N-BK7	AR	50.000	U
17	(aper)	Standard		-309.000	5.000			AR	50.000	U
18	(aper)	Standard		Infinity	5.000		N-BK7	AR	50.000	U
19	(aper)	Standard		-309.000	797.500			AR	50.000	U
20	(aper)	Standard		773.000	5.000		N-BK7	AR	50.000	U
21	(aper)	Standard		Infinity	40.000				50.000	U
22	(aper)	Standard		Infinity	5.500		SUPRASIL		48.000	U
23	(aper)	Standard		Infinity	640.446	V			50.000	U
24	(aper and tilts)	Standard		Infinity	-768.400		MIRROR		100.000	U
25	IMAGE (tilt/de	Standard		Infinity	-				42.399	

Table B.1: Detailed specification of the various distances and parameters necessary to generate the optical system for the Phantom v710 case. The radius column refers to the radius of curvature of the optical elements, the thickness the distance separation between two faces of those elements, and the semi-diameter their radial size, all of which are in mm. The materials were selected to have the same optical qualities as those that would be actually used and blank materials indicate air. These values have been optimized within Zemax to provide the best focus at the detection plane. The overall distance between the gas puff and the detection plane is approximately 2800 mm.

	Surf:Type		Comment	Radius	Thickness		Material	Coating	Semi-Diameter	
0	OBJECT (aper)	Standard		Infinity	750.769				23.162	
1	(aper)	Standard		Infinity	5.000		N-BK7	AR	50.000	U
2	(aper)	Standard		-773.000	5.000			AR	50.000	U
3	(aper)	Standard		Infinity	5.000		N-BK7	AR	50.000	U
4	(aper)	Standard		-773.000	568.700			AR	50.000	U
5	(aper)	Standard		773.000	5.000		N-BK7	AR	50.000	U
6	(aper)	Standard		Infinity	20.000				50.000	U
7	STOP	Standard		Infinity	20.000				45.000	U
8	(aper)	Standard		Infinity	5.500		SUPRASIL		48.000	U
9	(aper)	Standard		Infinity	688.811	V			50.000	U
10	(aper and tilts)	Standard		Infinity	-720.300		MIRROR		85.000	U
11	IMAGE (tilt/dec)	Standard		Infinity	-				47.812	

Table B.2: Detailed specification of the various distances and parameters necessary to generate the optical system for the APDCAM-10G case. The columns are defined the same way as those in Table B.1. These values have been optimized within Zemax to provide the best focus at the detection plane. The overall distance between the gas puff and the detection plane is also approximately 2800 mm.

Appendix C

Code and Scripts

C.1 block_design.tcl

```
source $board_path/config/ports.tcl

# Add PS and AXI Interconnect
set board_preset $board_path/config/board_preset.tcl
source $sdk_path/fpga/lib/starting_point.tcl

# Add ADCs and DACs
source $sdk_path/fpga/lib/redp_adc_dac.tcl
set adc_dac_name adc_dac
add_redp_adc_dac $adc_dac_name

# Rename clocks
set adc_clk $adc_dac_name/adc_clk

# Add processor system reset synchronous to adc clock
set rst_adc_clk_name proc_sys_reset_adc_clk
cell xilinx.com:ip:proc_sys_reset:5.0 $rst_adc_clk_name {} {
  ext_reset_in $ps_name/FCLK_RESET0_N
  slowest_sync_clk $adc_clk
}

# Add config and status registers
source $sdk_path/fpga/lib/ctl_sts.tcl
add_ctl_sts $adc_clk $rst_adc_clk_name/peripheral_aresetn

# Connect LEDs
connect_port_pin led_o [get_slice_pin [ctl_pin led] 7 0]
```

```
# Connect ADC to status register
for {set i 0} {$i < [get_parameter n_adc]} {incr i} {
  connect_pins [sts_pin adc$i] adc_dac/adc[expr $i + 1]
}

## Add DAC controller
#source $sdk_path/fpga/lib/bram.tcl
#set dac_bram_name [add_bram dac]

#connect_pins adc_dac/dac1 [get_slice_pin $dac_bram_name/doutb 13 0]
#connect_pins adc_dac/dac2 [get_slice_pin $dac_bram_name/doutb 29 16]

#connect_cell $dac_bram_name {
#  web  [get_constant_pin 0 4]
#  dinb [get_constant_pin 0 32]
#  clkb $adc_clk
 # rstb $rst_adc_clk_name/peripheral_reset
#}


# Use AXI Stream clock converter (ADC clock -> FPGA clock)
set intercon_idx 0
set idx [add_master_interface $intercon_idx]
cell xilinx.com:ip:axis_clock_converter:1.1 adc_clock_converter {
  TDATA_NUM_BYTES 4
} {
  s_axis_aresetn $rst_adc_clk_name/peripheral_aresetn
  m_axis_aresetn [set rst${intercon_idx}_name]/peripheral_aresetn
  s_axis_aclk $adc_clk
  m_axis_aclk [set ps_clk$intercon_idx]
}

# Add AXI stream FIFO to read pulse data from the PS
cell xilinx.com:ip:axi_fifo_mm_s:4.1 adc_axis_fifo {
  C_USE_TX_DATA 0
  C_USE_TX_CTRL 0
  C_USE_RX_CUT_THROUGH true
  C_RX_FIFO_DEPTH 32768
  C_RX_FIFO_PF_THRESHOLD 32760
} {
  s_axi_aclk [set ps_clk$intercon_idx]
  s_axi_aresetn [set rst${intercon_idx}_name]/peripheral_aresetn
  S_AXI [set interconnect_${intercon_idx}_name]/M${idx}_AXI
  AXI_STR_RXD adc_clock_converter/M_AXIS
```

```
}

assign_bd_address [get_bd_addr_segs adc_axis_fifo/S_AXI/Mem0]
set memory_segment  [get_bd_addr_segs /${::ps_name}/Data/SEG_adc_axis_fifo_Mem0]
set_property offset [get_memory_offset adc_fifo] $memory_segment
set_property range  [get_memory_range adc_fifo]  $memory_segment

################################## Make all the Blocks ####################################
create_bd_cell -type ip -vlnv PSFC:user:data_collector:1.0 data_collector_0
create_bd_cell -type ip -vlnv xilinx.com:ip:xlconstant:1.1 xlconstant_0

################################## Connected all the Blocks ###############################

connect_bd_net [get_bd_pins data_collector_0/tdata]
    [get_bd_pins adc_clock_converter/s_axis_tdata]
connect_bd_net [get_bd_pins data_collector_0/tvalid]
    [get_bd_pins adc_clock_converter/s_axis_tvalid]
connect_bd_net [get_bd_pins data_collector_0/adc_clk] [get_bd_pins adc_dac/adc_clk]
connect_bd_net [get_bd_pins xlconstant_0/dout] [get_bd_pins data_collector_0/clk_en]


# Connect ADC pins to pressure gauge status registers
connect_bd_net [get_bd_pins adc_dac/adc1] [get_bd_pins sts/abs_gauge]
    -boundary_type upper
connect_bd_net [get_bd_pins adc_dac/adc2] [get_bd_pins sts/diff_gauge]
    -boundary_type upper

# Add pins from extension connectors
create_bd_port -dir I W7X_permission_in
create_bd_port -dir I W7X_timings
create_bd_port -dir O -from 0 -to 0 slow_1_trigger
create_bd_port -dir O -from 0 -to 0 slow_2_trigger
create_bd_port -dir O -from 0 -to 0 slow_3_trigger
create_bd_port -dir O -from 0 -to 0 slow_4_trigger
create_bd_port -dir O -from 0 -to 0 fast_1_trigger
create_bd_port -dir O -from 0 -to 0 fast_1_permission_1
create_bd_port -dir O -from 0 -to 0 fast_1_duration_1
create_bd_port -dir O -from 0 -to 0 fast_1_permission_2
create_bd_port -dir O -from 0 -to 0 fast_1_duration_2
create_bd_port -dir O -from 0 -to 0 fast_2_trigger
create_bd_port -dir O -from 0 -to 0 fast_2_permission_1
create_bd_port -dir O -from 0 -to 0 fast_2_duration_1
create_bd_port -dir O -from 0 -to 0 fast_2_permission_2
create_bd_port -dir O -from 0 -to 0 fast_2_duration_2
```

```
# Add block which processes the outputs
create_bd_cell -type ip -vlnv PSFC:user:outputs:1.0 outputs_0

# Connect output pins
connect_bd_net [get_bd_pins adc_dac/adc_clk]
    [get_bd_pins outputs_0/adc_clk]
connect_bd_net [get_bd_pins ctl/GPI_safe_state]
    [get_bd_pins outputs_0/GPI_safe_state_ctl]
connect_bd_net [get_bd_pins ctl/slow_1_trigger]
    [get_bd_pins outputs_0/slow_1_trigger_ctl]
connect_bd_net [get_bd_pins ctl/slow_2_trigger]
    [get_bd_pins outputs_0/slow_2_trigger_ctl]
connect_bd_net [get_bd_pins ctl/slow_3_trigger]
    [get_bd_pins outputs_0/slow_3_trigger_ctl]
connect_bd_net [get_bd_pins ctl/slow_4_trigger]
    [get_bd_pins outputs_0/slow_4_trigger_ctl]
connect_bd_net [get_bd_ports W7X_permission_in]
    [get_bd_pins outputs_0/W7X_permission_out_ctl]
connect_bd_net [get_bd_pins ctl/fast_1_trigger]
    [get_bd_pins outputs_0/fast_1_trigger_ctl]
connect_bd_net [get_bd_pins ctl/fast_1_permission_1]
    [get_bd_pins outputs_0/fast_1_permission_1_ctl]
connect_bd_net [get_bd_pins ctl/fast_1_duration_1]
    [get_bd_pins outputs_0/fast_1_duration_1_ctl]
connect_bd_net [get_bd_pins ctl/fast_1_permission_2]
    [get_bd_pins outputs_0/fast_1_permission_2_ctl]
connect_bd_net [get_bd_pins ctl/fast_1_duration_2]
    [get_bd_pins outputs_0/fast_1_duration_2_ctl]
connect_bd_net [get_bd_pins ctl/fast_2_trigger]
    [get_bd_pins outputs_0/fast_2_trigger_ctl]
connect_bd_net [get_bd_pins ctl/fast_2_permission_1]
    [get_bd_pins outputs_0/fast_2_permission_1_ctl]
connect_bd_net [get_bd_pins ctl/fast_2_duration_1]
    [get_bd_pins outputs_0/fast_2_duration_1_ctl]
connect_bd_net [get_bd_pins ctl/fast_2_permission_2]
    [get_bd_pins outputs_0/fast_2_permission_2_ctl]
connect_bd_net [get_bd_pins ctl/fast_2_duration_2]
    [get_bd_pins outputs_0/fast_2_duration_2_ctl]

connect_bd_net [get_bd_ports slow_1_trigger]
    [get_bd_pins outputs_0/slow_1_trigger_pin]
connect_bd_net [get_bd_ports slow_2_trigger]
    [get_bd_pins outputs_0/slow_2_trigger_pin]
connect_bd_net [get_bd_ports slow_3_trigger]
    [get_bd_pins outputs_0/slow_3_trigger_pin]
```

```
connect_bd_net [get_bd_ports slow_4_trigger]
    [get_bd_pins outputs_0/slow_4_trigger_pin]
connect_bd_net [get_bd_ports fast_1_trigger]
    [get_bd_pins outputs_0/fast_1_trigger_pin]
connect_bd_net [get_bd_ports fast_1_permission_1]
    [get_bd_pins outputs_0/fast_1_permission_1_pin]
connect_bd_net [get_bd_ports fast_1_duration_1]
    [get_bd_pins outputs_0/fast_1_duration_1_pin]
connect_bd_net [get_bd_ports fast_1_permission_2]
    [get_bd_pins outputs_0/fast_1_permission_2_pin]
connect_bd_net [get_bd_ports fast_1_duration_2]
    [get_bd_pins outputs_0/fast_1_duration_2_pin]
connect_bd_net [get_bd_ports fast_2_trigger]
    [get_bd_pins outputs_0/fast_2_trigger_pin]
connect_bd_net [get_bd_ports fast_2_permission_1]
    [get_bd_pins outputs_0/fast_2_permission_1_pin]
connect_bd_net [get_bd_ports fast_2_duration_1]
    [get_bd_pins outputs_0/fast_2_duration_1_pin]
connect_bd_net [get_bd_ports fast_2_permission_2]
    [get_bd_pins outputs_0/fast_2_permission_2_pin]
connect_bd_net [get_bd_ports fast_2_duration_2]
    [get_bd_pins outputs_0/fast_2_duration_2_pin]

connect_bd_net [get_bd_pins outputs_0/GPI_safe_state_pin] [get_bd_pins adc_dac/dac1]
connect_bd_net [get_bd_pins outputs_0/W7X_permission_out_pin]
    [get_bd_pins adc_dac/dac2]
```

C.2 config.yml

```
---
name: GPI_2
board: boards/red-pitaya
version: 0.1.1

cores:
  - fpga/cores/redp_adc_v1_0
  - fpga/cores/redp_dac_v1_0
  - fpga/cores/axi_ctl_register_v1_0
  - fpga/cores/axi_sts_register_v1_0
  - fpga/cores/dna_reader_v1_0
  - instruments/GPI_2/cores/data_collector_v1_0
  - instruments/GPI_2/cores/outputs_v1_0

memory:
  - name: control
    offset: '0x60000000'
    range: 4K
  - name: status
    offset: '0x50000000'
    range: 4K
  - name: dac
    offset: '0x40000000'
    range: 32K
  - name: adc_fifo
    offset: '0x43C10000'
    range: 64K

control_registers:
  - led
  - GPI_safe_state
  - slow_1_trigger
  - slow_2_trigger
  - slow_3_trigger
  - slow_4_trigger
  - fast_1_trigger
  - fast_1_permission_1
  - fast_1_duration_1
  - fast_1_permission_2
  - fast_1_duration_2
  - fast_2_trigger
  - fast_2_permission_1
  - fast_2_duration_1
```

```
  - fast_2_permission_2
  - fast_2_duration_2


status_registers:
  - adc[n_adc]
  - W7X_permission_in
  - W7X_timings
  - abs_gauge
  - diff_gauge

parameters:
  fclk0: 166666667
  bram_addr_width: 13
  dac_width: 14
  adc_width: 14
  n_adc: 2

xdc:
  - boards/red-pitaya/config/ports.xdc
  - boards/red-pitaya/config/clocks.xdc
  - instruments/GPI_2/extension_connector.xdc


drivers:
  - ./GPI_2.hpp
  - server/drivers/common.hpp

# web:
#   - web/koheron.ts
#   - web/jquery.flot.d.ts
#   - ./web/pulse_generator.ts
#   - ./web/app.ts
#   - ./web/control.ts
#   - ./web/plot.ts
#   - ./web/index.html
#   - web/main.css
#   - web/navigation.ts
```

C.3 extension_connector.xdc

```
### Adding all of the necessary pins from the extension connectors


set_property IOSTANDARD LVCMOS33 [get_ports W7X_permission_in]
set_property IOSTANDARD LVCMOS33 [get_ports W7X_timings]

set_property PACKAGE_PIN G17 [get_ports W7X_permission_in]
set_property PACKAGE_PIN H16 [get_ports W7X_timings]




set_property IOSTANDARD LVCMOS33 [get_ports {slow_1_trigger[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {slow_2_trigger[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {slow_3_trigger[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {slow_4_trigger[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {fast_1_trigger[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {fast_1_permission_1[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {fast_1_duration_1[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {fast_1_permission_2[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {fast_1_duration_2[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {fast_2_trigger[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {fast_2_permission_1[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {fast_2_duration_1[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {fast_2_permission_2[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {fast_2_duration_2[0]}]



set_property PACKAGE_PIN J18 [get_ports {slow_1_trigger[0]}]
set_property PACKAGE_PIN K17 [get_ports {slow_2_trigger[0]}]
set_property PACKAGE_PIN L14 [get_ports {slow_3_trigger[0]}]
set_property PACKAGE_PIN L16 [get_ports {slow_4_trigger[0]}]
set_property PACKAGE_PIN G18 [get_ports {fast_1_trigger[0]}]
set_property PACKAGE_PIN H17 [get_ports {fast_1_permission_1[0]}]
set_property PACKAGE_PIN H18 [get_ports {fast_1_duration_1[0]}]
set_property PACKAGE_PIN K18 [get_ports {fast_1_permission_2[0]}]
set_property PACKAGE_PIN L15 [get_ports {fast_1_duration_2[0]}]
set_property PACKAGE_PIN L17 [get_ports {fast_2_trigger[0]}]
set_property PACKAGE_PIN J16 [get_ports {fast_2_permission_1[0]}]
set_property PACKAGE_PIN M15 [get_ports {fast_2_duration_1[0]}]
set_property PACKAGE_PIN K16 [get_ports {fast_2_permission_2[0]}]
set_property PACKAGE_PIN M14 [get_ports {fast_2_duration_2[0]}]
```

C.4 GPI_2.hpp

```
/// GPI_2 driver
///
/// (c) Koheron

#ifndef __DRIVERS_GPI_2_HPP__
#define __DRIVERS_GPI_2_HPP__

#include <atomic>
#include <thread>
#include <chrono>

#include <context.hpp>

// http://www.xilinx.com/support/documentation/ip_documentation/
    axi_fifo_mm_s/v4_1/pg080-axi-fifo-mm-s.pdf
namespace Fifo_regs {
    constexpr uint32_t rdfr = 0x18;
    constexpr uint32_t rdfo = 0x1C;
    constexpr uint32_t rdfd = 0x20;
    constexpr uint32_t rlr = 0x24;
}

constexpr uint32_t dac_size = mem::dac_range/sizeof(uint32_t);
constexpr uint32_t adc_buff_size = 16777216;

class GPI_2
{
  public:
    GPI_2(Context& ctx_)
    : ctx(ctx_)
    , ctl(ctx.mm.get<mem::control>())
    , sts(ctx.mm.get<mem::status>())
    , adc_fifo_map(ctx.mm.get<mem::adc_fifo>())
    , adc_data(adc_buff_size)
    // , dac_map(ctx.mm.get<mem::dac>())
    {
        start_fifo_acquisition();
    }

    // GPI_2 generator
    void set_led(uint32_t led) {
        ctl.write<reg::led>(led);
    }
```

```
// Adc FIFO

uint32_t get_fifo_occupancy() {
   return adc_fifo_map.read<Fifo_regs::rdfo>();
}

void reset_fifo() {
    adc_fifo_map.write<Fifo_regs::rdfr>(0x000000A5);
}

uint32_t read_fifo() {
    return adc_fifo_map.read<Fifo_regs::rdfd>();
}

uint32_t get_fifo_length() {
   return (adc_fifo_map.read<Fifo_regs::rlr>() & 0x3FFFFF) >> 2;
}

 // Function to return the buffer length
uint32_t get_buffer_length() {
   return collected;
}

void set_GPI_safe_state(uint32_t state) {
    ctl.write<reg::GPI_safe_state>(state);
}

uint32_t get_GPI_safe_state() {
    return ctl.read<reg::GPI_safe_state>();
}

void set_slow_1_trigger(uint32_t state) {
    ctl.write<reg::slow_1_trigger>(state);
}

uint32_t get_slow_1_trigger() {
    return ctl.read<reg::slow_1_trigger>();
}

void set_slow_2_trigger(uint32_t state) {
    ctl.write<reg::slow_2_trigger>(state);
}

uint32_t get_slow_2_trigger() {
```

```
    return ctl.read<reg::slow_2_trigger>();
}

void set_slow_3_trigger(uint32_t state) {
    ctl.write<reg::slow_3_trigger>(state);
}

uint32_t get_slow_3_trigger() {
    return ctl.read<reg::slow_3_trigger>();
}

void set_slow_4_trigger(uint32_t state) {
    ctl.write<reg::slow_4_trigger>(state);
}

uint32_t get_slow_4_trigger() {
    return ctl.read<reg::slow_4_trigger>();
}

void set_fast_1_trigger(uint32_t state) {
    ctl.write<reg::fast_1_trigger>(state);
}

uint32_t get_fast_1_trigger() {
    return ctl.read<reg::fast_1_trigger>();
}

void set_fast_1_permission_1(uint32_t state) {
    ctl.write<reg::fast_1_permission_1>(state);
}

uint32_t get_fast_1_permission_1() {
    return ctl.read<reg::fast_1_permission_1>();
}

void set_fast_1_duration_1(uint32_t state) {
    ctl.write<reg::fast_1_duration_1>(state);
}

uint32_t get_fast_1_duration_1() {
    return ctl.read<reg::fast_1_duration_1>();
}

void set_fast_1_permission_2(uint32_t state) {
    ctl.write<reg::fast_1_permission_2>(state);
```

```
}

uint32_t get_fast_1_permission_2() {
    return ctl.read<reg::fast_1_permission_2>();
}

void set_fast_1_duration_2(uint32_t state) {
    ctl.write<reg::fast_1_duration_2>(state);
}

uint32_t get_fast_1_duration_2() {
    return ctl.read<reg::fast_1_duration_2>();
}

void set_fast_2_trigger(uint32_t state) {
    ctl.write<reg::fast_2_trigger>(state);
}

uint32_t get_fast_2_trigger() {
    return ctl.read<reg::fast_2_trigger>();
}

void set_fast_2_permission_1(uint32_t state) {
    ctl.write<reg::fast_2_permission_1>(state);
}

uint32_t get_fast_2_permission_1() {
    return ctl.read<reg::fast_2_permission_1>();
}

void set_fast_2_duration_1(uint32_t state) {
    ctl.write<reg::fast_2_duration_1>(state);
}

uint32_t get_fast_2_duration_1() {
    return ctl.read<reg::fast_2_duration_1>();
}

void set_fast_2_permission_2(uint32_t state) {
    ctl.write<reg::fast_2_permission_2>(state);
}

uint32_t get_fast_2_permission_2() {
    return ctl.read<reg::fast_2_permission_2>();
}
```

```
  void set_fast_2_duration_2(uint32_t state) {
      ctl.write<reg::fast_2_duration_2>(state);
  }

  uint32_t get_fast_2_duration_2() {
      return ctl.read<reg::fast_2_duration_2>();
  }

  uint32_t get_W7X_permission() {
      return sts.read<reg::W7X_permission_in>();
  }

  uint32_t get_abs_gauge() {
      return sts.read<reg::abs_gauge>();
  }

  uint32_t get_diff_gauge() {
      return sts.read<reg::diff_gauge>();
  }

   // Function to return data
std::vector<uint32_t>& get_GPI_2_data() {

  if (dataAvailable){
    dataAvailable = false;

    return adc_data;
  } else {
    return empty_vector;
  }
}

  // void wait_for(uint32_t n_pts) {
  //     do {} while (get_fifo_length() < n_pts);
  // }



  void start_fifo_acquisition();

private:
  Context& ctx;
  Memory<mem::control>& ctl;
  Memory<mem::status>& sts;
```

```
    Memory<mem::adc_fifo>& adc_fifo_map;
    // Memory<mem::dac>& dac_map;

    std::vector<uint32_t> adc_data;
    std::vector<uint32_t> empty_vector;

    uint32_t fill_buffer(uint32_t);

    std::atomic<bool> fifo_acquisition_started{false};
    std::atomic<bool> dataAvailable{false};
    std::atomic<uint32_t> collected{0};          //number of currently collected data

    std::thread fifo_thread;
    void fifo_acquisition_thread();

};

inline void GPI_2::start_fifo_acquisition() {
    if (! fifo_acquisition_started) {
        // fifo_buffer.fill(0);
        fifo_thread = std::thread{&GPI_2::fifo_acquisition_thread, this};
        fifo_thread.detach();
    }
}

inline void GPI_2::fifo_acquisition_thread() {
  constexpr auto fifo_sleep_for = std::chrono::nanoseconds(5000);
  fifo_acquisition_started = true;
  ctx.log<INFO>("Starting fifo acquisition");
  adc_data.reserve(16777216);
  adc_data.resize(0);
  empty_vector.resize(0);

  uint32_t dropped=0;

  // While loop to reserve the number of samples needed to be collected
  while (fifo_acquisition_started){
    // if (dataAvailable == true) {
    //   ctx.log<INFO>("Reached checkpoint alpha");
    // } else {
    //   ctx.log<INFO>("Reached checkpoint charlie");
    // }
    if (collected == 0){
      // Checking that data has not yet been collected
      if ((dataAvailable == false) && (adc_data.size() > 0)){
```

```
    // Sleep to avoid a race condition while data is being transferred
    std::this_thread::sleep_for(fifo_sleep_for);
    // Clearing vector back to zero
    adc_data.resize(0);
    ctx.log<INFO>("vector cleared, adc_data size: %d", adc_data.size());
      }
    }

    dropped = fill_buffer(dropped);
    if (dropped > 0){
      ctx.log<INFO>("Dropped samples: %d", dropped);
    }
    // std::this_thread::sleep_for(fifo_sleep_for);
  }// While loop
}

// Member function to fill buffer array
inline uint32_t GPI_2::fill_buffer(uint32_t dropped) {
    // Retrieving the number of samples to collect
  uint32_t samples=get_fifo_length();
  //ctx.log<INFO>("Samples: %d", samples);

  // Collecting samples in buffer
  if (samples > 0) {
    // Checking for dropped samples
    if (samples >= 32768){
      dropped += 1;
    }
    for (size_t i=0; i < samples; i++){
      adc_data.push_back(read_fifo());
      collected = collected + 1;
    }
  }
  // if statement for setting the acquisition completed flag
  if (samples == 0) {
    if (collected > 0) {
      dataAvailable = true;
      collected = 0;
      dropped = 0;
    }
  }
  return dropped;
}
```

```
#endif // __DRIVERS_PCS_HPP__
```

C.5 GPI_2.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import time
import math
import numpy as np

from koheron import command

class GPI_2(object):
    def __init__(self, client):
        self.client = client
        # self.n_pts = 16384
        self.n_pts = 8192
        self.fs = 125e6 # sampling frequency (Hz)

        self.adc = np.zeros((2, self.n_pts))
        self.dac = np.zeros((2, self.n_pts))


    @command()
    def set_led(self, led):
        pass

    @command()
    def get_fifo_length(self):
        return self.client.recv_uint32()

    @command()
    def get_buffer_length(self):
        return self.client.recv_uint32()

    @command()
    def get_Template_data(self):
        return self.client.recv_vector(dtype='uint32')

    @command()
    def set_GPI_safe_state(self, state):
        pass

    @command()
    def get_GPI_safe_state(self):
        return self.client.recv_uint32()
```

```
@command()
def set_slow_1_trigger(self, state):
    pass

@command()
def get_slow_1_trigger(self):
    return self.client.recv_uint32()

@command()
def set_slow_2_trigger(self, state):
    pass

@command()
def get_slow_2_trigger(self):
    return self.client.recv_uint32()

@command()
def set_slow_3_trigger(self, state):
    pass

@command()
def get_slow_3_trigger(self):
    return self.client.recv_uint32()

@command()
def set_slow_4_trigger(self, state):
    pass

@command()
def get_slow_4_trigger(self):
    return self.client.recv_uint32()

@command()
def set_fast_1_trigger(self, state):
    pass

@command()
def get_fast_1_trigger(self):
    return self.client.recv_uint32()

@command()
def set_fast_1_permission_1(self, state):
    pass
```

```
@command()
def get_fast_1_permission_1(self):
    return self.client.recv_uint32()

@command()
def set_fast_1_duration_1(self, state):
    pass

@command()
def get_fast_1_duration_1(self):
    return self.client.recv_uint32()

@command()
def set_fast_1_permission_2(self, state):
    pass

@command()
def get_fast_1_permission_2(self):
    return self.client.recv_uint32()

@command()
def set_fast_1_duration_2(self, state):
    pass

@command()
def get_fast_1_duration_2(self):
    return self.client.recv_uint32()

@command()
def set_fast_2_trigger(self, state):
    pass

@command()
def get_fast_2_trigger(self):
    return self.client.recv_uint32()

@command()
def set_fast_2_permission_1(self, state):
    pass

@command()
def get_fast_2_permission_1(self):
    return self.client.recv_uint32()

@command()
```

```
    def set_fast_2_duration_1(self, state):
        pass

    @command()
    def get_fast_2_duration_1(self):
        return self.client.recv_uint32()

    @command()
    def set_fast_2_permission_2(self, state):
        pass

    @command()
    def get_fast_2_permission_2(self):
        return self.client.recv_uint32()

    @command()
    def set_fast_2_duration_2(self, state):
        pass

    @command()
    def get_fast_2_duration_2(self):
        return self.client.recv_uint32()

    @command()
    def get_W7X_permission(self):
        return self.client.recv_uint32()

    @command()
    def get_abs_gauge(self):
        return self.client.recv_uint32()

    @command()
    def get_diff_gauge(self):
        return self.client.recv_uint32()
```

C.6 DataCollect.vhd

```
----------------------------------------------------------------------------------
-- Module to organise and store data for the MLP project
-- Started on March 26th by Charlie Vincent
--
-- Adjust variable is to lengthen period to a number that is indivisible by three
-- First two levels will be of length period, third level will be of length
-- period + adjust
----------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity DataCollect is
  port (
    adc_clk    : in std_logic;          -- adc input clock
    data_1_valid : in std_logic;
    data_2_valid : in std_logic;
    data_1       : in std_logic_vector(15 downto 0);
    data_2       : in std_logic_vector(15 downto 0);
    clk_en     : in std_logic;

    tvalid : out std_logic;
    tdata  : out std_logic_vector(31 downto 0)
    );

end entity DataCollect;

architecture Behavioral of DataCollect is


begin  -- architecture Behavioral

  -- purpose: Process to collect voltage values
  -- type   : combinational
  -- inputs : adc_clk
  -- outputs: data
  temp_collect : process (adc_clk) is
  begin  -- process data_collect
    if rising_edge(adc_clk) then
      if clk_en = '1' then
        if data_1_valid = '1' then
```

```
          tdata <= data_1 & data_2;

          tvalid <= '1';
        end if;
    end if;
  end if;
  end process temp_collect;

end architecture Behavioral;
```

C.7 outputs.vhd

```
----------------------------------------------------------------------------------
-- Intermediate block between RP control registers and output pins
----------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity outputs is

    port (
        adc_clk                 : in std_logic;
        GPI_safe_state_ctl      : in std_logic_vector(31 downto 0);
        slow_1_trigger_ctl      : in std_logic_vector(31 downto 0);
        slow_2_trigger_ctl      : in std_logic_vector(31 downto 0);
        slow_3_trigger_ctl      : in std_logic_vector(31 downto 0);
        slow_4_trigger_ctl      : in std_logic_vector(31 downto 0);
        W7X_permission_out_ctl  : in std_logic_vector(11 downto 0);
        fast_1_trigger_ctl      : in std_logic_vector(31 downto 0);
        fast_1_permission_1_ctl : in std_logic_vector(31 downto 0);
        fast_1_duration_1_ctl   : in std_logic_vector(31 downto 0);
        fast_1_permission_2_ctl : in std_logic_vector(31 downto 0);
        fast_1_duration_2_ctl   : in std_logic_vector(31 downto 0);
        fast_2_trigger_ctl      : in std_logic_vector(31 downto 0);
        fast_2_permission_1_ctl : in std_logic_vector(31 downto 0);
        fast_2_duration_1_ctl   : in std_logic_vector(31 downto 0);
        fast_2_permission_2_ctl : in std_logic_vector(31 downto 0);
        fast_2_duration_2_ctl   : in std_logic_vector(31 downto 0);


        GPI_safe_state_pin      : out std_logic;
        slow_1_trigger_pin      : out std_logic;
        slow_2_trigger_pin      : out std_logic;
        slow_3_trigger_pin      : out std_logic;
        slow_4_trigger_pin      : out std_logic;
        W7X_permission_out_pin  : out std_logic;
        fast_1_trigger_pin      : out std_logic;
        fast_1_permission_1_pin : out std_logic;
        fast_1_duration_1_pin   : out std_logic;
        fast_1_permission_2_pin : out std_logic;
        fast_1_duration_2_pin   : out std_logic;
        fast_2_trigger_pin      : out std_logic;
        fast_2_permission_1_pin : out std_logic;
```

```
        fast_2_duration_1_pin    : out std_logic;
        fast_2_permission_2_pin : out std_logic;
        fast_2_duration_2_pin    : out std_logic
    );

end entity outputs;

architecture Behavioral of outputs is

begin

    Pass_through_0 : process(adc_clk)
    begin
        if (rising_edge(adc_clk)) then
            if (GPI_safe_state_ctl /= "00000000000000000000000000000000") then
                GPI_safe_state_pin <= '1';
            else
                GPI_safe_state_pin <= '0';
            end if;
        end if;
    end process Pass_through_0;

    Pass_through_1 : process(adc_clk)
    begin
        if (rising_edge(adc_clk)) then
            if (slow_1_trigger_ctl /= "00000000000000000000000000000000") then
                slow_1_trigger_pin <= '1';
            else
                slow_1_trigger_pin <= '0';
            end if;
        end if;
    end process Pass_through_1;

    Pass_through_2 : process(adc_clk)
    begin
        if (rising_edge(adc_clk)) then
            if (slow_2_trigger_ctl /= "00000000000000000000000000000000") then
                slow_2_trigger_pin <= '1';
            else
                slow_2_trigger_pin <= '0';
            end if;
        end if;
    end process Pass_through_2;

    Pass_through_3 : process(adc_clk)
```

```
begin
    if (rising_edge(adc_clk)) then
        if (slow_3_trigger_ctl /= "00000000000000000000000000000000") then
            slow_3_trigger_pin <= '1';
        else
            slow_3_trigger_pin <= '0';
        end if;
    end if;
end process Pass_through_3;

Pass_through_4 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
        if (slow_4_trigger_ctl /= "00000000000000000000000000000000") then
            slow_4_trigger_pin <= '1';
        else
            slow_4_trigger_pin <= '0';
        end if;
    end if;
end process Pass_through_4;

Pass_through_5 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
        if (W7X_permission_out_ctl = "000000000000") then
            W7X_permission_out_pin <= '0';
        else
            W7X_permission_out_pin <= '1';
        end if;
    end if;
end process Pass_through_5;

Pass_through_6 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
        if (fast_1_trigger_ctl /= "00000000000000000000000000000000") then
            fast_1_trigger_pin <= '1';
        else
            fast_1_trigger_pin <= '0';
        end if;
    end if;
end process Pass_through_6;

Pass_through_7 : process(adc_clk)
begin
```

```
    if (rising_edge(adc_clk)) then
        if (fast_1_permission_1_ctl /= "00000000000000000000000000000000") then
            fast_1_permission_1_pin <= '1';
        else
            fast_1_permission_1_pin <= '0';
        end if;
    end if;
end process Pass_through_7;

Pass_through_8 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
        if (fast_1_duration_1_ctl /= "00000000000000000000000000000000") then
            fast_1_duration_1_pin <= '1';
        else
            fast_1_duration_1_pin <= '0';
        end if;
    end if;
end process Pass_through_8;

Pass_through_9 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
        if (fast_1_permission_2_ctl /= "00000000000000000000000000000000") then
            fast_1_permission_2_pin <= '1';
        else
            fast_1_permission_2_pin <= '0';
        end if;
    end if;
end process Pass_through_9;

Pass_through_10 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
        if (fast_1_duration_2_ctl /= "00000000000000000000000000000000") then
            fast_1_duration_2_pin <= '1';
        else
            fast_1_duration_2_pin <= '0';
        end if;
    end if;
end process Pass_through_10;

Pass_through_11 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
```

```
        if (fast_2_trigger_ctl /= "00000000000000000000000000000000") then
            fast_2_trigger_pin <= '1';
        else
            fast_2_trigger_pin <= '0';
        end if;
    end if;
end process Pass_through_11;

Pass_through_12 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
        if (fast_2_permission_1_ctl /= "00000000000000000000000000000000") then
            fast_2_permission_1_pin <= '1';
        else
            fast_2_permission_1_pin <= '0';
        end if;
    end if;
end process Pass_through_12;

Pass_through_13 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
        if (fast_2_duration_1_ctl /= "00000000000000000000000000000000") then
            fast_2_duration_1_pin <= '1';
        else
            fast_2_duration_1_pin <= '0';
        end if;
    end if;
end process Pass_through_13;

Pass_through_14 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
        if (fast_2_permission_2_ctl /= "00000000000000000000000000000000") then
            fast_2_permission_2_pin <= '1';
        else
            fast_2_permission_2_pin <= '0';
        end if;
    end if;
end process Pass_through_14;

Pass_through_15 : process(adc_clk)
begin
    if (rising_edge(adc_clk)) then
        if (fast_2_duration_2_ctl /= "00000000000000000000000000000000") then
```

```
                fast_2_duration_2_pin <= '1';
            else
                fast_2_duration_2_pin <= '0';
            end if;
        end if;
    end process Pass_through_15;


end architecture Behavioral;
```

C.8 GPI_Valve_GUI.py

```
from __future__ import print_function # for print to work inside lambda
from Tkinter import *
from PIL import Image, ImageTk
from matplotlib import pyplot as plt
import sys
import time
import numpy as np
import os
import csv

from GPI_2 import GPI_2

from koheron import connect

GPI_host = os.getenv('HOST', 'w7xrp1')
GPI_client = connect(GPI_host, name='GPI_2')
GPI_driver = GPI_2(GPI_client)

root = Tk()
root.title("GPI Valve Control")

win_width = 1450
win_height = 880
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
x = (screen_width / 2) - (win_width / 2)
y = (screen_height / 2) - (win_height / 2)
root.geometry('%dx%d+%d+%d' % (win_width, win_height, x, y))
root.columnconfigure(4, weight=1)
root.columnconfigure(5, weight=1)
root.rowconfigure(10, weight=1)
root.rowconfigure(11, weight=1)
root.rowconfigure(12, weight=1)

image = Image.open("background.png")
photo = ImageTk.PhotoImage(image)
background = Label(image=photo)
background.image = photo
background.grid(rowspan=10, columnspan=4)

fast_valve_indicator = Canvas(root,width=29, height=43)
fast_valve_indicator.place(x=68, y=210)
fast_valve_status = fast_valve_indicator.create_rectangle(0, 0, 29, 43)
```

```
fast_valve_indicator.itemconfig(fast_valve_status, fill="red")
# def get_fast_status():
#     if (GPI_driver.get_fast_1_trigger() == 0 or
#         GPI_driver.get_fast_1_permission_1() == 0 or
#         GPI_driver.get_fast_1_duration_1() == 0 or
#         GPI_driver.get_fast_1_permission_2() == 0 or
#         GPI_driver.get_fast_1_duration_2() == 0):
#         fast_valve_indicator.itemconfig(fast_valve_status, fill="red")
#     else:
#         fast_valve_indicator.itemconfig(fast_valve_status, fill="green")
# get_fast_valve_status()
fast_valve_label_back = Label(text="FV2", width=13)
fast_valve_label_back.place(x=125, y=125)
def fast_valve_open():
    fast_win = Toplevel()
    fast_win_width = 250
    fast_win_height = 75
    fast_x = (screen_width / 2) - (fast_win_width / 2)
    fast_y = (screen_height / 2) - (fast_win_height / 2)
    fast_win.geometry("%dx%d+%d+%d" %
                      (fast_win_width, fast_win_height, fast_x, fast_y))
    fast_win.title("FV2")
    fast_win.rowconfigure(0, weight=1)
    fast_win.rowconfigure(1, weight=1)
    fast_win.columnconfigure(0, weight=1)
    fast_win.columnconfigure(1, weight=1)
    msg = Message(fast_win, text="Please confirm the OPENING of the FV2.", width=200)
    msg.grid(columnspan=2)
    def fast_confirm():
        GPI_driver.set_fast_1_trigger(1)
#         get_fast_status()
        fast_valve_indicator.itemconfig(fast_valve_status, fill="green")
        fast_win.destroy()
    confirm = Button(fast_win, text="Confirm", width=10, command=fast_confirm)
    confirm.grid(row=1, column=0)
    cancel = Button(fast_win, text="Cancel", width=10, command=fast_win.destroy)
    cancel.grid(row=1, column=1)
def fast_valve_close():
    fast_win = Toplevel()
    fast_win_width = 250
    fast_win_height = 75
    fast_x = (screen_width / 2) - (fast_win_width / 2)
    fast_y = (screen_height / 2) - (fast_win_height / 2)
    fast_win.geometry("%dx%d+%d+%d" %
                      (fast_win_width, fast_win_height, fast_x, fast_y))
```

```
    fast_win.title("FV2")
    fast_win.rowconfigure(0, weight=1)
    fast_win.rowconfigure(1, weight=1)
    fast_win.columnconfigure(0, weight=1)
    fast_win.columnconfigure(1, weight=1)
    msg = Message(fast_win, text="Please confirm the CLOSING of the FV2.", width=200)
    msg.grid(columnspan=2)
    def fast_confirm():
        GPI_driver.set_fast_1_trigger(0)
#         get_fast_status()
        fast_valve_indicator.itemconfig(fast_valve_status, fill="red")
        fast_win.destroy()
    confirm = Button(fast_win, text="Confirm", width=10, command=fast_confirm)
    confirm.grid(row=1, column=0)
    cancel = Button(fast_win, text="Cancel", width=10, command=fast_win.destroy)
    cancel.grid(row=1, column=1)
fast_valve_open_button = Button(root, text="OPEN", fg="green", width=10,
                                command=fast_valve_open)
fast_valve_open_button.place(x=125, y=150)
fast_valve_close_button = Button(root, text="CLOSE", fg="red", width=10,
                                 command=fast_valve_close)
fast_valve_close_button.place(x=125, y=180)

slow_valve_1_indicator = Canvas(root,width=29, height=43)
slow_valve_1_indicator.place(x=417, y=464)
slow_valve_1_status = slow_valve_1_indicator.create_rectangle(0, 0, 29, 43)
slow_valve_1_indicator.itemconfig(slow_valve_1_status, fill="red")
# def get_slow_1_status():
#     if GPI_driver.get_slow_1_trigger() == 0:
#         slow_valve_1_indicator.itemconfig(slow_valve_1_status, fill="red")
#     else:
#         slow_valve_1_indicator.itemconfig(slow_valve_1_status, fill="green")
# get_slow_1_status()
slow_valve_1_label_back = Label(text="V5", width=13)
slow_valve_1_label_back.place(x=475, y=380)
def slow_valve_1_open():
    slow_1_win = Toplevel()
    slow_1_win_width = 250
    slow_1_win_height = 75
    slow_1_x = (screen_width / 2) - (slow_1_win_width / 2)
    slow_1_y = (screen_height / 2) - (slow_1_win_height / 2)
    slow_1_win.geometry("%dx%d+%d+%d" % (slow_1_win_width, slow_1_win_height,
                        slow_1_x, slow_1_y))
    slow_1_win.title("V5")
    slow_1_win.rowconfigure(0, weight=1)
```

```
    slow_1_win.rowconfigure(1, weight=1)
    slow_1_win.columnconfigure(0, weight=1)
    slow_1_win.columnconfigure(1, weight=1)
    msg = Message(slow_1_win, text="Please confirm the OPENING of V5.", width=200)
    msg.grid(columnspan=2)
    def slow_1_confirm():
        GPI_driver.set_slow_1_trigger(1)
#         get_slow_1_status()
        slow_valve_1_indicator.itemconfig(slow_valve_1_status, fill="green")
        slow_1_win.destroy()
    confirm = Button(slow_1_win, text="Confirm", width=10, command=slow_1_confirm)
    confirm.grid(row=1, column=0)
    cancel = Button(slow_1_win, text="Cancel", width=10, command=slow_1_win.destroy)
    cancel.grid(row=1, column=1)
def slow_valve_1_close():
    slow_1_win = Toplevel()
    slow_1_win_width = 250
    slow_1_win_height = 75
    slow_1_x = (screen_width / 2) - (slow_1_win_width / 2)
    slow_1_y = (screen_height / 2) - (slow_1_win_height / 2)
    slow_1_win.geometry("%dx%d+%d+%d" % (slow_1_win_width, slow_1_win_height,
                        slow_1_x, slow_1_y))
    slow_1_win.title("V5")
    slow_1_win.rowconfigure(0, weight=1)
    slow_1_win.rowconfigure(1, weight=1)
    slow_1_win.columnconfigure(0, weight=1)
    slow_1_win.columnconfigure(1, weight=1)
    msg = Message(slow_1_win, text="Please confirm the CLOSING of V5.", width=200)
    msg.grid(columnspan=2)
    def slow_1_confirm():
        GPI_driver.set_slow_1_trigger(0)
#         get_slow_1_status()
        slow_valve_1_indicator.itemconfig(slow_valve_1_status, fill="red")
        slow_1_win.destroy()
    confirm = Button(slow_1_win, text="Confirm", width=10, command=slow_1_confirm)
    confirm.grid(row=1, column=0)
    cancel = Button(slow_1_win, text="Cancel", width=10, command=slow_1_win.destroy)
    cancel.grid(row=1, column=1)
slow_valve_1_open_button = Button(root, text="OPEN", fg="green", width=10,
                                  command=slow_valve_1_open)
slow_valve_1_open_button.place(x=475, y=405)
slow_valve_1_close_button = Button(root, text="CLOSE", fg="red", width=10,
                                   command=slow_valve_1_close)
slow_valve_1_close_button.place(x=475, y=435)
```

```
slow_valve_2_indicator = Canvas(root,width=43, height=29)
slow_valve_2_indicator.place(x=509, y=571)
slow_valve_2_status = slow_valve_2_indicator.create_rectangle(0, 0, 43, 29)
slow_valve_2_indicator.itemconfig(slow_valve_2_status, fill="red")
# def get_slow_2_status():
#     if GPI_driver.get_slow_2_trigger() == 0:
#         slow_valve_2_indicator.itemconfig(slow_valve_2_status, fill="red")
#     else:
#         slow_valve_2_indicator.itemconfig(slow_valve_2_status, fill="green")
# get_slow_2_status()
slow_valve_2_label_back = Label(text="V4", width=13)
slow_valve_2_label_back.place(x=310, y=545)
def slow_valve_2_open():
    slow_2_win = Toplevel()
    slow_2_win_width = 250
    slow_2_win_height = 75
    slow_2_x = (screen_width / 2) - (slow_2_win_width / 2)
    slow_2_y = (screen_height / 2) - (slow_2_win_height / 2)
    slow_2_win.geometry("%dx%d+%d+%d" % (slow_2_win_width, slow_2_win_height,
                        slow_2_x, slow_2_y))
    slow_2_win.title("V4")
    slow_2_win.rowconfigure(0, weight=1)
    slow_2_win.rowconfigure(1, weight=1)
    slow_2_win.columnconfigure(0, weight=1)
    slow_2_win.columnconfigure(1, weight=1)
    msg = Message(slow_2_win, text="Please confirm the OPENING of V4.", width=200)
    msg.grid(columnspan=2)
    def slow_2_confirm():
        GPI_driver.set_slow_2_trigger(1)
        #get_slow_2_status()
        slow_valve_2_indicator.itemconfig(slow_valve_2_status, fill="green")
        slow_2_win.destroy()
    confirm = Button(slow_2_win, text="Confirm", width=10, command=slow_2_confirm)
    confirm.grid(row=1, column=0)
    cancel = Button(slow_2_win, text="Cancel", width=10, command=slow_2_win.destroy)
    cancel.grid(row=1, column=1)
def slow_valve_2_close():
    slow_2_win = Toplevel()
    slow_2_win_width = 250
    slow_2_win_height = 75
    slow_2_x = (screen_width / 2) - (slow_2_win_width / 2)
    slow_2_y = (screen_height / 2) - (slow_2_win_height / 2)
    slow_2_win.geometry("%dx%d+%d+%d" % (slow_2_win_width, slow_2_win_height,
                        slow_2_x, slow_2_y))
    slow_2_win.title("V4")
```

```
    slow_2_win.rowconfigure(0, weight=1)
    slow_2_win.rowconfigure(1, weight=1)
    slow_2_win.columnconfigure(0, weight=1)
    slow_2_win.columnconfigure(1, weight=1)
    msg = Message(slow_2_win, text="Please confirm the CLOSING of V4.", width=200)
    msg.grid(columnspan=2)
    def slow_2_confirm():
        GPI_driver.set_slow_2_trigger(0)
#         get_slow_2_status()
        slow_valve_2_indicator.itemconfig(slow_valve_2_status, fill="red")
        slow_2_win.destroy()
    confirm = Button(slow_2_win, text="Confirm", width=10, command=slow_2_confirm)
    confirm.grid(row=1, column=0)
    cancel = Button(slow_2_win, text="Cancel", width=10, command=slow_2_win.destroy)
    cancel.grid(row=1, column=1)
slow_valve_2_open_button = Button(root, text="OPEN", fg="green", width=10,
                                  command=slow_valve_2_open)
slow_valve_2_open_button.place(x=310, y=570)
slow_valve_2_close_button = Button(root, text="CLOSE", fg="red", width=10,
                                   command=slow_valve_2_close)
slow_valve_2_close_button.place(x=310, y=600)

slow_valve_3_indicator = Canvas(root,width=43, height=29)
slow_valve_3_indicator.place(x=661, y=374)
slow_valve_3_status = slow_valve_3_indicator.create_rectangle(0, 0, 43, 29)
slow_valve_3_indicator.itemconfig(slow_valve_3_status, fill="green")
# def get_slow_3_status():
#     if GPI_driver.get_slow_3_trigger() == 0:
#         slow_valve_3_indicator.itemconfig(slow_valve_3_status, fill="green")
#     else:
#         slow_valve_3_indicator.itemconfig(slow_valve_3_status, fill="red")
# get_slow_3_status()
slow_valve_3_label_back = Label(text="V3", width=13)
slow_valve_3_label_back.place(x=795, y=345)
def slow_valve_3_open():
    slow_3_win = Toplevel()
    slow_3_win_width = 250
    slow_3_win_height = 75
    slow_3_x = (screen_width / 2) - (slow_3_win_width / 2)
    slow_3_y = (screen_height / 2) - (slow_3_win_height / 2)
    slow_3_win.geometry("%dx%d+%d+%d" % (slow_3_win_width, slow_3_win_height,
                        slow_3_x, slow_3_y))
    slow_3_win.title("V3")
    slow_3_win.rowconfigure(0, weight=1)
    slow_3_win.rowconfigure(1, weight=1)
```

```
    slow_3_win.columnconfigure(0, weight=1)
    slow_3_win.columnconfigure(1, weight=1)
    msg = Message(slow_3_win, text="Please confirm the OPENING of V3.", width=200)
    msg.grid(columnspan=2)
    def slow_3_confirm():
        GPI_driver.set_slow_3_trigger(1)
#         get_slow_3_status()
        slow_valve_3_indicator.itemconfig(slow_valve_3_status, fill="green")
        slow_3_win.destroy()
    confirm = Button(slow_3_win, text="Confirm", width=10, command=slow_3_confirm)
    confirm.grid(row=1, column=0)
    cancel = Button(slow_3_win, text="Cancel", width=10, command=slow_3_win.destroy)
    cancel.grid(row=1, column=1)
def slow_valve_3_close():
    slow_3_win = Toplevel()
    slow_3_win_width = 250
    slow_3_win_height = 75
    slow_3_x = (screen_width / 2) - (slow_3_win_width / 2)
    slow_3_y = (screen_height / 2) - (slow_3_win_height / 2)
    slow_3_win.geometry("%dx%d+%d+%d" % (slow_3_win_width, slow_3_win_height,
                        slow_3_x, slow_3_y))
    slow_3_win.title("V3")
    slow_3_win.rowconfigure(0, weight=1)
    slow_3_win.rowconfigure(1, weight=1)
    slow_3_win.columnconfigure(0, weight=1)
    slow_3_win.columnconfigure(1, weight=1)
    msg = Message(slow_3_win, text="Please confirm the CLOSING of V3.", width=200)
    msg.grid(columnspan=2)
    def slow_3_confirm():
        GPI_driver.set_slow_3_trigger(0)
#         get_slow_3_status()
        slow_valve_3_indicator.itemconfig(slow_valve_3_status, fill="red")
        slow_3_win.destroy()
    confirm = Button(slow_3_win, text="Confirm", width=10, command=slow_3_confirm)
    confirm.grid(row=1, column=0)
    cancel = Button(slow_3_win, text="Cancel", width=10, command=slow_3_win.destroy)
    cancel.grid(row=1, column=1)
slow_valve_3_open_button = Button(root, text="OPEN", fg="green", width=10,
                                  command=slow_valve_3_open)
slow_valve_3_open_button.place(x=795, y=370)
slow_valve_3_close_button = Button(root, text="CLOSE", fg="red", width=10,
                                   command=slow_valve_3_close)
slow_valve_3_close_button.place(x=795, y=400)
```

```
abs_gauge_label_back = Label(text="Absolute Pressure Gauge")
abs_gauge_label_back.place(x=605, y=5)
diff_gauge_label_back = Label(text="Differential Pressure Gauge")
diff_gauge_label_back.place(x=855, y=170)

abs_gauge_label = Label(text="Absolute Pressure Gauge Reading:")
abs_gauge_label.grid(row=0, column=4, columnspan=2)
abs_gauge_graph = Canvas(root, width=200, height=200, background="grey")
abs_gauge_graph.grid(row=1, column=4, columnspan=2)

diff_gauge_label = Label(text="Differential Pressure Gauge Reading:")
diff_gauge_label.grid(row=3, column=4, columnspan=2)
diff_gauge_graph = Canvas(root, width=200, height=200, background="grey")
diff_gauge_graph.grid(row=4, column=4, columnspan=2)

desired_pressure_label = Label(text="Desired Pressure:")
desired_pressure_label.grid(row=6, column=4)
desired_pressure_entry = Entry(root, width=10)
desired_pressure_entry.grid(row=6, column=5)

fill_button = Button(root, text="Fill", width=10)
fill_button.grid(row=7, column=4)

pump_refill_button = Button(root, text="Pump & Refill", width=10)
pump_refill_button.grid(row=7, column=5)

local_permission_1_label = Label(text="Local Permission #1")
local_permission_1_label.grid(row=10, column=0)
timing_1_label = Label(text="FV2 Opening Timing #1")
timing_1_label.grid(row=11, column=0)
duration_1_label = Label(text="Opening Duration #1")
duration_1_label.grid(row=12, column=0)

local_permission_1_var = IntVar()
def print_1_check():
    if local_permission_1_var.get():
        GPI_driver.set_fast_1_permission_1(1)
    else:
        GPI_driver.set_fast_1_permission_1(0)
local_permission_1_check = Checkbutton(root, variable=local_permission_1_var,
                                       command=print_1_check)
local_permission_1_check.grid(row=10, column=1)
timing_1_entry = Entry(root, width=10)
def calc_clock_cycles(event):
    cycles_in_entry = int(int(timing_1_entry.get())/8e-9)
```

```
    cycles_in_duration = int(int(duration_1_entry.get())/8e-9)
    print(cycles_in_entry)
    print(cycles_in_duration)
timing_1_entry.bind("<Return>", calc_clock_cycles)
timing_1_entry.grid(row=11, column=1)
duration_1_entry = Entry(root, width=10)
duration_1_entry.grid(row=12, column=1)
duration_1_entry.bind("<Return>", calc_clock_cycles)

local_permission_2_label = Label(text="Local Permission #2")
local_permission_2_label.grid(row=10, column=2)
timing_2_label = Label(text="FV2 Opening Timing #2")
timing_2_label.grid(row=11, column=2)
duration_2_label = Label(text="Opening Duration #2")
duration_2_label.grid(row=12, column=2)

local_permission_2_var = IntVar()
def print_2_check():
    if local_permission_2_var.get():
        GPI_driver.set_fast_1_permission_2(1)
    else:
        GPI_driver.set_fast_1_permission_2(0)
local_permission_2_check = Checkbutton(root, variable=local_permission_2_var,
                                       command=print_2_check)
local_permission_2_check.grid(row=10, column=3)
timing_2_entry = Entry(root, width=10)
timing_2_entry.grid(row=11, column=3)
duration_2_entry = Entry(root, width=10)
duration_2_entry.grid(row=12, column=3)

W7X_permission_label = Label(text="W7-X Permission:")
W7X_permission_label.grid(row=10, column=4)
W7X_permission_status = Label(text="Granted/Forbidden")
W7X_permission_status.grid(row=10, column=5)

GPI_safe_state_label = Label(text="GPI Safe State:")
GPI_safe_state_label.grid(row=12, column=4)
GPI_safe_state_button = Button(root, text="ENABLE", width=10)
GPI_safe_state_button.grid(row=12, column=5)



root.mainloop()
```

Bibliography

[1] Yuhong Xu. A general comparison between tokamak and stellarator plasmas. *Matter and Radiation at Extremes*, 1(4):192–200, jul 2016.

[2] S. J. Zweben, J. L. Terry, D. P. Stotler, and R. J. Maqueda. Invited Review Article: Gas puff imaging diagnostics of edge plasma turbulence in magnetic fusion devices. *Review of Scientific Instruments*, 88(4):041101, apr 2017.

[3] 3.1.1. STEMlab 125-10 vs. STEMlab 125-14 (originally Red Pitaya v1.1) — Red Pitaya STEMlab 0.97 documentation, 2017.

[4] 3.1.3.2. Extension — Red Pitaya STEMlab 0.97 documentation, 2017.

[5] J. L. Terry, S. J. Zweben, K. Hallatschek, B. LaBombard, R. J. Maqueda, B. Bai, C. J. Boswell, M. Greenwald, D. Kopon, W. M. Nevins, C. S. Pitcher, B. N. Rogers, D. P. Stotler, and X. Q. Xu. Observations of the turbulence in the scrape-off-layer of Alcator C-Mod and comparisons with simulation. *Physics of Plasmas*, 10(5):1739–1747, may 2003.

[6] B. Yuan, M. Xu, Y. Yu, L. Zang, R. Hong, C. Chen, Z. Wang, L. Nie, R. Ke, D. Guo, Y. Wu, T. Long, S. Gong, H. Liu, M. Ye, X. Duan, and HL-2A Team. Development of a new gas puff imaging diagnostic on the HL-2A tokamak. *Journal of Instrumentation*, 13(03):C03033–C03033, mar 2018.

[7] R. J. Maqueda, G. A. Wurden, D. P. Stotler, S. J. Zweben, B. LaBombard, J. L. Terry, J. L. Lowrance, V. J. Mastrocola, G. F. Renda, D. A. D'Ippolito, J. R. Myra, and N. Nishino. Gas puff imaging of edge turbulence (invited). *Review of Scientific Instruments*, 74(3):2020–2026, mar 2003.

[8] K. Ikeda. Progress in the ITER Physics Basis. *Nuclear Fusion*, 47(6), jun 2007.

[9] Isabella Milch. Wendelstein 7-X fusion device produces its first hydrogen plasma, 2016.